Keith und Steven Brain

Commodore Sachbuchreihe Band 15

KÜNSTLICHE INTELLIGENZ AUF DEM COMMODORE 64

Wertvolle Ideen und Anwendungen
anwendbar auch auf dem Commodore 128

Commodore Sachbuchreihe Band 15

Keith und Steven Brain

KÜNSTLICHE INTELLIGENZ AUF DEM COMMODORE 64



Commodore

Titel der Originalausgabe: Keith & Steven Brain – Artificial Intelligence on the Commodore 64

First published 1984 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street, London WC2R 3LD

Aus dem Englischen übertragen von Wolfgang Dhein.

Artikel-Nr. 556431/04.85 Änderungen vorbehalten. ISBN-NR. 3-89133-015-4

INHALT

EINLEITUNG

Künstliche Intelligenz ist ohne jeden Zweifel in der Computerentwicklung ein Bereich von stetig wachsender Bedeutung, der in den nächsten Jahrzehnten einen starken Einfluß auf unser aller Leben haben wird. Sinn dieses Buches ist es, den Leser mit einigen Bereichen der künstlichen Intelligenz bekanntzumachen, und ihm zu zeigen, wie "intelligente" Routinen in BASIC entwickelt werden können, die er dann in seine eigenen Programme einfügen kann. Es wird nur eine oberflächliche Kenntnis von BASIC erwartet, und das Buch beginnt mit den Grundlagen, da wir der Meinung sind, daß man nur so die Probleme künstlicher Intelligenz verstehen und bewältigen kann.

Das Buch geht grundsätzlich davon aus, daß Ideen aufgegriffen werden und passende Routinen dafür Schritt für Schritt entwickelt, untersucht und wenn möglich alternative Lösungen gesucht werden. Anstatt nur eine Serie kompletter Programme anzugeben, ermutigen wir Sie, verschiedene Möglichkeiten auszuprobieren, um so eigene Resultate zu erhalten. Als weitere Hilfe haben wir detaillierte Ablaufdiagramme für die meisten Routinen beigefügt. Bei diesen Routinen liegt der Schwerpunkt bei den Aspekten der künstlichen Intelligenz; deshalb haben wir es vermieden, die Bildschirmdarstellung besonders zu "polieren", da dadurch der eigentliche Zweck des Programms in den Hintergrund tritt. In einigen Bereichen sind unnötig erscheinende Programmzeilen absichtlich eingefügt, um den Programmablauf klar darzustellen.

So weit wie möglich wurde die Wiederholung von Zeilen vermieden, aber deren Änderung ist im Laufe der weiteren Programmentwicklung normal. Alle Ausdrucke in diesem Buch haben das gleiche Format wie auf dem Bildschirm. In den meisten Fällen wurden Leerzeichen und Klammern benutzt, um die Ausdrucke lesbarer zu machen, was jedoch nicht bedeutet, daß sie nicht in einigen Fällen für den fehlerfreien Ablauf des Programms notwendig sind. Alle Routinen sind ausführlich getestet worden, und die Ausdrucke wurden sorgfältig überprüft, so daß wir hoffen, daß Sie keine Fehler finden werden. Es ist jedoch eine bekannte Tatsache, daß die meisten Fehler durch "Eingabefehler" des Benutzers entstehen. Semikolon und Komma sehen manchmal sehr unwichtig aus, aber ihr Fehlen kann schwere Folgen haben.

Künstliche Intelligenz wird von Tag zu Tag wichtiger, und wir hoffen, daß Ihnen dieses Buch einen nützlichen Einblick in diesen Bereich geben kann. Wer weiß – wenn Sie sich wirklich in diesen Bereich hineinknien, wird es Ihnen möglicherweise bald gelingen, Ihren Computer dazu zu überreden, daß er unser nächstes Buch selbst liest!

Keith und Steven Brain
Groeswen, im Januar 1984

KAPITEL 1

KÜNSTLICHE INTELLIGENZ

FANTASIE

Seit Generationen haben die Autoren von Zukunftsromanen die Entwicklung von intelligenten Maschinen vorausgesagt, die menschliche Aufgaben wahrnehmen (oder sie sogar in manchen Bereichen übertreffen) können, und haben damit unbestreitbar die öffentliche Meinung über künstliche Intelligenz geprägt. Daraus ergab sich die landläufige Meinung, daß ein Roboter eine intelligente Maschine mit allgemein menschlichem Aussehen ist, die selbständig ihm allgemein gegebene Anweisungen ausführen kann.

In den meisten Fällen haben Menschen unterschwellige Angstgefühle in bezug auf die Technik, so daß in den ersten Zukunftsromanen diese Roboter meistens die Rolle der fast unbesiegbaren und gewissenlosen "Bösen" übernehmen mußten. Der weitsichtige Isaac Asimov schrieb eine lange Reihe von Geschichten über diese "Positronic Roboter" und war wahrscheinlich der erste Autor, der mit den Wirklichkeiten dieser Situation umgehen konnte. Er stellte die "Drei Gesetze der Automation" auf, die die Grundregeln für jede Maschine bilden, die zu unabhängigen Aktionen fähig sein soll. Interessanterweise war er jedoch nicht in der Lage vorauszusehen, wann die Menschheit die Anwesenheit solcher Maschinen auf der Erde akzeptieren würde.

"Der Krieg der Sterne" kennt die spezialisierten Roboter R2D2 und C3P0, die jedoch mit vielen ihrer Eigenschaften auf uns einen sehr seltsamen Eindruck machten. Vielleicht hat eine interplanetarische Robotergewerkschaft die direkte Kommunikation zwischen Menschen und R2D2 vertraglich verhindert. In "Die Ehefrauen von Stepford" hatten deren Männer die (gute?) Idee, ihre Ehefrauen in "Androiden" zu verwandeln, die automatisch genau das taten, was man von ihnen erwartete. Aber die Fortsetzung zeigte die Gefahren der Notwendigkeit, das Verhalten der Androiden laufend erneut von außen bestimmen zu müssen. Hoffentlich haben eventuelle außerirdische Eindringlinge nicht den "Kampfstern Galactica" gesehen und werden deshalb Roboter des Typs "CYLON" bauen, die wie die alten "Space Invaders" letztlich wegen ihrer völligen Durchschaubarkeit immer vernichtet werden konnten.

Natürlich gibt es intelligente Computer auch in Gehäusen ohne Arme und Beine, obwohl flackernde Lämpchen obligatorisch zu sein scheinen. Die Kommunikation mit ihnen ist natürlich verbal, aber die Tage der alten metallischen Stimme sind vorbei, und eine gewisse Persönlichkeit ist ihnen nicht mehr abzusprechen. Es wäre

eine gute Idee, sie alle gleich aussehen zu lassen, aber lassen Sie Ihren bitte nicht wie Sergeant Zero in "Terrahawks" sprechen! Mit Michael Knight's KITT kann man sich schon recht angenehm unterhalten, und er ist auf jeden Fall dem schwierigen SLAVE und dem widerwärtigen ORAC aus "Blake's Sieben" vorzuziehen. Durch ORAC enthält diese kleine Plexiglasbox eine enorme Portion von Verachtung, aber dadurch werden wahrscheinlich die Probleme vermieden, die bei einem zu nahen Abbild des Menschen durch eine Maschine entstehen könnten.

Der superintelligente Computer HAL in Arthur C. Clarke's Spielfilm "2001 Odyssee im Weltraum" bekam einen Nervenzusammenbruch, als er mit zu viel Verantwortung konfrontiert wurde; die intelligente Bombe in "Der dunkle Stern" diskutierte begeistert mit Kapitän Doolittle über den Existentialismus, aber war nicht bereit, von der vorgesehenen Explosionszeit abzuweichen, obwohl sie noch im Bombenschacht war. In "Das Restaurant am Ende des Universums" wurde der "Glückliche Menschentransporter" der Sirius Cybernetics Gesellschaft wertlos, als er sich weigerte zu starten, als er in die Zukunft sehen konnte und dort seine wahrscheinliche Zerstörung voraussah, und der "Nutri-Matic Getränkeautomat" ist wahrscheinlich von der Deutschen Schlafwagengesellschaft entwickelt worden, da er immer ein Getränk produzierte, das nicht im geringsten etwas mit Kaffee zu tun hatte.

Beängstigendere Dinge passierten in der jüngsten Vergangenheit. Das wichtigste Geschehen des Filmes "Kriegsspiele" war nicht die Tatsache, daß es jemand gelang, in JOSHUA (den Verteidigungs-Computer der USA) einzudringen, sondern, daß die Maschine, einmal angefangen einen Atomkrieg zu spielen, dieses nicht mehr stoppte, bis jemand das Spiel gewonnen hatte. In dem Film „Das Forbin Projekt" taten sich die amerikanischen und russischen Computer zusammen und waren sich darüber einig, daß die Menschen sowieso unnötig sind. Wenn Sie natürlich "Marvin – der paranoide Android" sind und ein Gehirn von der Größe eines Planeten und eine eigene menschliche Persönlichkeit haben, dann können Sie die feindliche Maschine auch ohne Waffen besiegen, indem Sie ihr den Boden unter den Füßen durch die Diskussion der eigenen Probleme wegziehen.

WIRKLICHKEIT

Die Definition und das Erkennen maschineller Intelligenz wird zwischen den entsprechenden Experten heftig diskutiert. Die am meisten anerkannte Definition wurde zuerst von Alan Turing in den späten 40er Jahren entwickelt, als Computer so groß wie Häuser und seltener als Rechenschieber heute waren. Anstatt eine Reihe zu erfüllender Kriterien zu definieren, betrachtete er das Problem aus einer weiteren Perspektive. Er erkannte, daß die meisten Menschen davon ausgehen, daß die meisten anderen Menschen intelligent sind, und wenn ein Mensch nicht unterscheiden kann, ob er es mit einem anderen Menschen oder nur mit einem

Computer zu tun hat, er diese Maschine dann als intelligent anerkennt. Diese Tatsache bildet die Grundlage des bekannten "Turing Tests", bei dem die Testperson eine Unterhaltung mit einem "anderen" über eine Tastatur führen muß und dabei versuchen muß herauszubekommen, ob der andere eine Maschine oder auch ein Mensch ist.
Viele Geschichten handeln von diesem Test, aber wir finden die am interessantesten, bei der ein Bewerber um einen Arbeitsplatz einfach an eine Tastatur gesetzt und sich dort überlassen wird. Natürlich erkennt er die Wichtigkeit dieses Tests für seine berufliche Zukunft und versucht mit Gewalt das Geheimnis zu lüften, offensichtlich ohne Erfolg. Nach einiger Zeit kommt der Interviewer jedoch zurück, schüttelt ihm die Hand und gratuliert ihm mit den Wörtern: "Gut gemacht, die Maschine konnte nicht feststellen, ob Sie ein Mensch sind. Sie sind also für die Position als Steuerbeamter bestens geeignet".
Jedermann weiß aus der Fernsehwerbung, daß computerunterstützte Entwicklung inzwischen zu den normalen Techniken gehört, und daß sich in den Werkshallen von Autofabriken fast nur noch Industrieroboter aufhalten, und an den Autofenstern Aufkleber mit der Aussage "Entwickelt von einem Computer, hergestellt von einem Roboter und gefahren von einem Idioten" befinden. In Wirklichkeit haben die meisten dieser Industrieroboter nur eine minimale Intelligenz, da sie lediglich vorgegebenen Arbeitsprozessen folgen, ohne wirkliche Entscheidungen zu treffen. Sogar der eindrucksvolle Lackierautomat folgt nur gläubig den Bewegungen, die ein Mensch vorher mit seinem Arm durchgeführt hat. Er ist nicht in der Lage, neue Teile ohne weitere neue Unterstützung durch einen Menschen zu lackieren.
Andererseits wird die nächste Generation dieser Roboter sehr hoch entwickelte Sensoren und Software haben, mit denen sie die Umrisse, Farben und Oberflächen von Objekten erkennen können, und die ihnen erlauben werden, rationale Entscheidungen zu treffen. Jeder, der die Berichte über die legendären "Micromouse"-Wettbewerbe gesehen hat, bei dem die elektrischen Nager selbständig und zielstrebig (?) dem Zentrum eines Irrgartens zustreben, kann nicht mehr überrascht sein, wenn wir an die Zukunft von intelligenten Robotern glauben, obwohl es wahrscheinlich keinen Sinn hat, ihnen zwei Arme und zwei Beine zu geben.
Ein anderer wichtiger Bereich, in dem zur Zeit künstliche Intelligenz benutzt wird, sind die "Expertensysteme". Viele von ihnen vollführen ihre Arbeit genau so gut oder sogar schon besser als menschliche Experten, speziell im Bereich der Wettervoraussage. Diese Systeme können für viele Bereiche benutzt werden, im Speziellen sind sie von ständig wachsender Bedeutung in den Bereichen der medizinischen Diagnose und Behandlung – jedoch brauchen sich die Ärzte keine Sorgen um ihre Zukunft zu machen, da die Computer bestimmt niemals in der Lage sein werden, die Patienten zu verhätscheln.
Das größte Problem in bezug auf eine breitere Benutzung von Computern ist die Unwissenheit und Engstirnigkeit der Benutzer, die die Bedienungsanleitungen nur

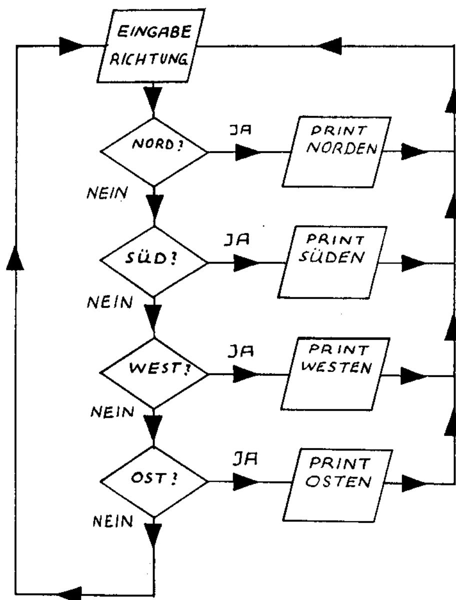
als letzte Rettung lesen und von der Maschine erwarten, daß diese all ihre kleinen Absonderlichkeiten versteht. Die Verarbeitung des "gesprochenen Wortes" ist deshalb ein Bereich, in dem noch viel getan werden muß, und die Computer der fünften Generation werden sehr viel benutzerfreundlicher sein.
In den meisten Bereichen der künstlichen Intelligenz werden besser passende (aber auch exotischere) Programmiersprachen als BASIC, z. B. LISP und PROLOG benutzt, die für den normalen Benutzer ziemlich unverständlich sind und wahrscheinlich nicht auf Ihrem Mirkocomputer zu Hause zur Verfügung stehen. Von den in den nächsten Kapiteln folgenden BASIC-Routinen können Sie natürlich nicht erwarten, daß sie Ihnen alle Möglichkeiten der künstlichen Intelligenz eröffnen. Sie werden Ihnen jedoch ein ziemlich gutes Gefühl für die Möglichkeiten und Probleme der künstlichen Intelligenz vermitteln.

KAPITEL 2

NUR BEFEHLEN FOLGEN

Da Ihr Computer in Wirklichkeit ziemlich unintelligent ist, können Sie sich mit ihm nur in sehr einfacher Weise unterhalten. Der erste Schritt, der in vielen einfachen Abenteuerspielen verwendet wird, besteht aus einer Reihe von vorgegebenen Befehlen, für die es fest vorgegebene Antworten gibt. Wir wollen damit beginnen, daß wir nach vorgegebenen Himmelsrichtungen suchen, um uns in eine bestimmte Richtung zu bewegen. Am einfachsten scheint es zu sein, in dem Programm nach einem INPUT des Benutzers zu fragen und dann für jede einzelne Möglichkeit eine getrennte IF-THEN-Zeile zu schreiben (siehe **Ablaufdiagramm 2.1**).

```
100 PRINT"RICHTUNG ?";
120 INPUT IN$
200 IF IN$="NORD" THEN PRINT"NORD"
210 IF IN$="SUED" THEN PRINT"SUED"
220 IF IN$="WEST" THEN PRINT"WEST"
230 IF IN$="OST"  THEN PRINT"OST"
250 GOTO 100
```

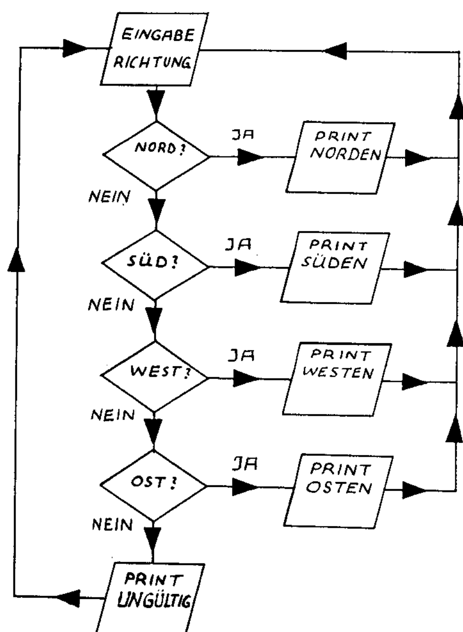


Ablaufdiagramm 2.1 Gibt Himmelsrichtungen an

Wenn nun irgend etwas anderes als eines der vier Schlüsselwörter eingegeben wird, wird nichts ausgedruckt, dafür aber eine neue Eingabe angefordert. Es wäre erheblich benutzerfreundlicher, wenn der Computer angeben würde, warum diese Eingabe nicht gültig war. Sie könnten das dadurch erreichen, daß ein Test eingefügt wird, der alle möglichen Schlüsselwörter abprüft. Bei einer großen Anzahl von gültigen Wörtern würde diese Vorgehensweise sehr umständlich und praktisch unmöglich werden.

```
240 IF IN$<>"NORD"AND IN$<>"SUED"AND IN$
<>"WEST"AND IN$<>"OST"THEN IN$=""
245 IF IN$=""THEN PRINT"UNGUELTIGE EINGA
BE":GOTO 100
```

Andererseits kann durch das Anfügen von GOTO 100 an das Ende jeder IF-THEN-Zeile ein direkter Rücksprung zum INPUT beim Erkennen eines gültigen Kommandos erzwungen werden. Wenn keiner der IF-Tests wahr ist, erreicht das Programm die Zeile 240, in der eine Warnung ausgedruckt wird. Es ist auf jeden Fall besser, nach dem Finden eines gültigen Wortes sofort zurückzuspringen, da es das System davor bewahrt, unnötige Tests bei bereits gefundener Antwort durchzuführen (siehe **Ablaufdiagramm 2.2**).



Ablaufdiagramm 2.2 Unnötige Tests vermeiden

```
200 IF IN$="NORD"THEN PRINT"NORD":GOTO 1
00
210 IF IN$="SUED"THEN PRINT"SUED":GOTO 1
00
220 IF IN$="WEST"THEN PRINT"WEST":GOTO 1
00
230 IF IN$="OST" THEN PRINT"OST" :GOTO 1
00
240 PRINT"UNGUELTIGE EINGABE"
```

In diesem Fall wird beim Auffinden eines gültigen Kommandos dieses nochmal auf dem Bildschirm angezeigt, aber ansonsten tut es nichts. Um den wirklichen Umgang mit den einzelnen Marschrichtungen zu lernen, definieren wir eine Startposition als X%=0 und Y%=0 und berücksichtigen die Bewegung relativ zu diesem Punkt durch plus und minus. Wenn immer möglich sollten Ganzzahlen (Integer) benutzt werden, da diese schneller als Realzahlen verarbeitet werden; außerdem wird dadurch die Kollision mit reservierten Namen vermindert.

```
10 X%=0:Y%=0
```

Wir müssen nun nur noch unsere Positionsbewegungen mit in den Befehl übernehmen und dem Benutzer mitteilen, daß wir ihn richtig verstanden haben (siehe **Ablaufdiagramm 2.3**).

```
200 IF IN$="NORD"THEN PRINT"NORD":Y%=Y%-
1:GOTO 100
210 IF IN$="SUED"THEN PRINT"SUED":Y%=Y%+
1:GOTO 100
220 IF IN$="WEST"THEN PRINT"WEST":X%=X%-
1:GOTO 100
230 IF IN$="OST" THEN PRINT"OST" :X%=X%+
1:GOTO 100
```

Durch diese Programmerweiterung wissen wir jetzt immer genau, wo wir uns in Bezug auf unseren Startpunkt befinden. Um nun auch noch auf dem Bildschirm zu zeigen, wo wir uns wirklich befinden, ist die folgende Befehlszeile einzufügen:

```
110 PRINT"X";X%,"Y";Y%
```

WIR BENUTZEN UNTERPROGRAMME

Das war bis jetzt ein sehr einfaches Beispiel. Wenn die Ergebnisse der Aktionen ein bißchen komplizierter werden, ist es im allgemeinen ratsam, die Antworten als Unterprogramme zu programmieren.

```
200 IF IN$="NORD"THEN GOSUB 2000:GOTO 10
0
210 IF IN$="SUED"THEN GOSUB 2100:GOTO 10
0
220 IF IN$="WEST"THEN GOSUB 2300:GOTO 10
0
230 IF IN$="OST" THEN GOSUB 2400:GOTO 10
0
2000 PRINT "NACH NORDEN":Y%=Y%-1:RETURN
2100 PRINT "NACH SUEDEN":Y%=Y%+1:RETURN
2200 PRINT "NACH WESTEN":X%=X%-1:RETURN
2300 PRINT "NACH OSTEN" :X%=X%+1:RETURN
```

MEHR FLEXIBILITÄT

Die Verwendung dieser IF-THEN-Tests könnte nun ins "Unendliche" erweitert werden; aber das wäre wirklich ein recht primitiver Weg, der nur Probleme bringt, wenn die Programme komplizierter werden sollen. Ein sehr viel eleganterer Weg, mit diesen Kommandowörtern und den entsprechenen Systemreaktionen zu arbeiten, kann durch ihre Eingabe über DATA und die Abspeicherung in Tabellen erreicht werden. Dazu müssen zuerst Tabellen entsprechender Länge für die Kommandowörter (C$) und die Antworten (R$) angelegt bzw. DIMensioniert werden. Da BASIC Zeichenketten variabler Länge zuläßt (bis zu 255 Zeichen), kann der aktuelle Text so gut wie jede Länge haben.

```
30 DIM C$(3),R$(3)
```

Wenn die Kommandos und die Antworten in den DATA-Anweisungen paarweise angeordnet werden, können sie nicht so leicht durcheinander kommen, und es ist sehr viel einfacher, sie in die entsprechenden Elemente jeder Tabelle einzulesen (siehe **Tabelle 2.1**).

```
10000 DATA NORD,NACH NORDEN,SUED,NACH SU
EDEN,WEST,NACH WESTEN,OST,NACH OSTEN
11000 FOR N=0 TO 3
11010 READ C$(N),R$(N)
11020 NEXT N
```

ELEMENT NUMMER	KOMMANDO WORT C$(N)	ANTWORT R$(N)
1	NORD	NACH NORDEN
2	SUED	NACH SUEDEN
3	WEST	NACH WESTEN
4	OST	NACH OSTEN

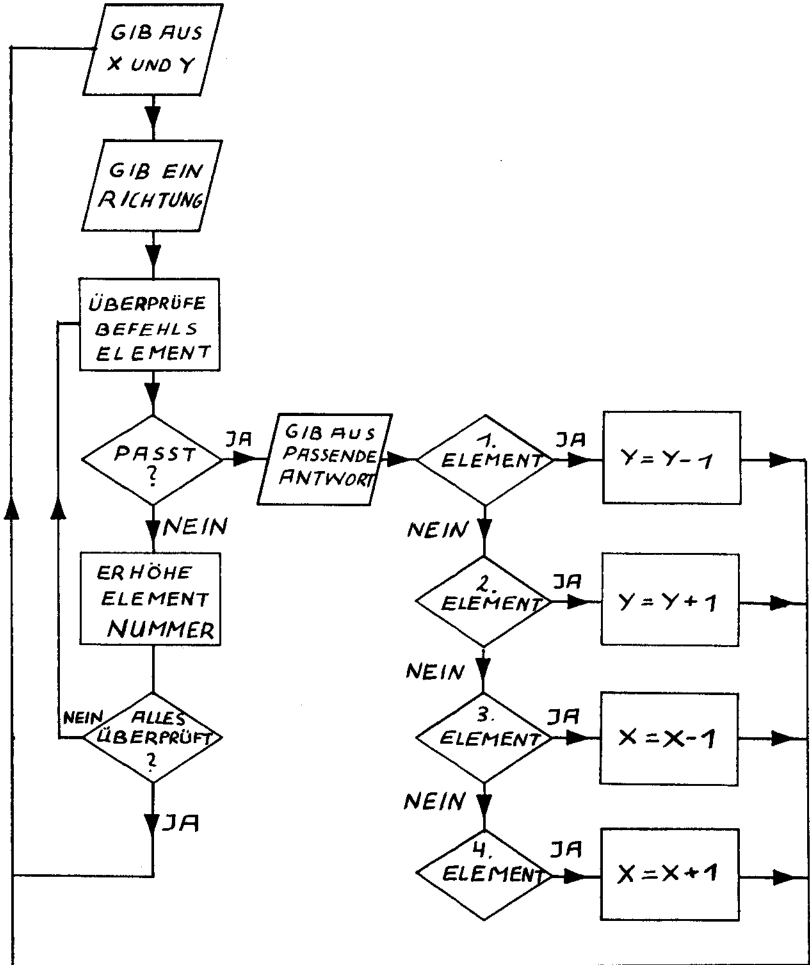
Tabelle 2.1 Inhalt der Kommando- und Antwortentabellen

Um die Tabellen während des Programmlaufs initialisieren zu können, das heißt, seine Elemente mit den Schlüsselwörtern und Antworten zu füllen, ist nur noch ein GOSUB und RETURN notwendig.

```
40 GOSUB10000
11030 RETURN
```

Diese umständlichen IF-THEN-Tests können nun durch eine einfache Schleife ersetzt werden, die den INPUT mit jedem Element der Tabelle, das die Kommandowörter (C$) enthält, vergleicht (siehe **Ablaufdiagramm 2.4**). Die Zeilen 200 bis 220 müssen durch die folgenden Zeilen ersetzt werden, und Zeile 230 muß gelöscht werden.

```
200 FOR N=0 TO 3
210 IF IN$=C$(N) THEN PRINT R$(N):GOTO 1
00
220 NEXT N
```



Ablaufdiagramm 2.4 Mehr Beweglichkeit

Falls die Eingabe IN$ mit einem der Kommandowörter übereinstimmt, verläßt das Programm die Suchschleife und gibt die passende Antwort R$(N) aus.
Natürlich sind wir immer noch bei der Startposition, da wir noch keine Bewegung durchgeführt haben. Es muß also noch sichergestellt werden, daß die Unterprogramme aufgerufen werden, mit denen die jeweiligen Bewegungen durchgeführt werden. Zuerst müssen wir jedoch den Sprung aus der Suchschleife für den Fall bewerkstelligen, daß unser Schlüsselwort gefunden worden ist.

```
210 IF IN$=C$(N) THEN PRINT R$(N):GOTO 3
00
```

Da wir den Wert von N (der Nummer des letzten überprüften Tabellenelementes) kennen, wissen wir auch, bei welchem Schlüsselwort wir die Übereinstimmung mit der Eingabe hatten. Wir können diesen Wert in einer ON-GOSUB-Anweisung zur Auswahl des richtigen Unterprogramms benutzen. Diese Unterprogramme ähneln denen, die wir bereits vorher geschrieben haben, aber es ist nicht mehr nötig, die entsprechende Antwort auszugeben, da dies bereits in Zeile 210 geschehen ist.

```
300 ON(N+1) GOSUB 2000,2100,2200,2300:GO
TO 100
2000 Y%=Y%-1:RETURN
2100 Y%=Y%+1:RETURN
2200 X%=X%-1:RETURN
2300 X%=X%+1:RETURN
```

WIR ERWEITERN DAS VOKABULAR

Die Tabellen können sehr leicht erweitert werden, um mehr Wörter aufnehmen zu können. Zum Erreichen einer größtmöglichen Flexibilität dimensionieren wir die Tabellen mit der Variablen WD%, die wir ebenfalls für die Einlese- und die Suchschleife benutzen. Dadurch bekommen wir eine allgemein gültige Routine, die leicht durch Zuweisung neuer Werte für die Variable WD% geändert werden kann.

```
20 WD%=3
30 DIM C$(WD%),R$(WD%)
200 FOR N=0 TO WD%
11000 FOR N=0 TO WD%
```

Z. B. können wir zusätzliche Richtungen der Kompaßrose benutzen,

```
20 WD%=7
10010 DATA NORDOST,NACH NORDOSTEN,SUEDOS
T,NACH SUEDOSTEN
10020 DATA SUEDWEST,NACH SUEDWESTEN,NORD
WEST,NACH NORDWESTEN
```

für die wir natürlich zusätzliche Unterprogramme benötigen:

```
300 ON(N+1) GOSUB 2000,2100,2200,2300,24
00,2500,2600,2700:GOTO 100

2400 Y%=Y%-1:X%=X%+1:RETURN
2500 Y%=Y%+1:X%=X%+1:RETURN
2600 Y%=Y%+1:X%=X%-1:RETURN
2700 Y%=Y%-1:X%=X%-1:RETURN
```

WIR BEREINIGEN DAS PROGRAMM

Alle Antworten, die wir bis jetzt benutzt haben, haben das Wort "nach" benutzt, das wir deshalb in jeder DATA-Anweisung eingeben mußten. Diese Tippübungen mögen zwar für einen angehenden Sekretär sehr praktisch sein, aber für einen Programmierer gibt es einen eleganteren Weg; er definiert sich dieses vielbenutzte Wort ebenfalls als Zeichenkette. Es darf dabei nicht vergessen werden, am Ende des Wortes ein Leerzeichen anzufügen, um es von den folgenden Wörtern abtrennen zu können.

```
10100 G$="NACH "
```

Jetzt ist endlich die Möglichkeit gegeben, dieses Wort in allen DATA-Anweisungen zu löschen. Dafür muß natürlich die Variable G$ mit jedem Schlüsselwort in der Antwort ausgegeben werden.

```
210 IF IN$=C$(N) THEN PRINT G$;R$(N):GOT
O 300
10000 DATA NORD,NORDEN,SUED,SUEDEN,WEST,
WESTEN,OST,OSTEN
10010 DATA NORDOST,NORDOSTEN,SUEDOST,SUE
DOSTEN
10020 DATA SUEDWEST,SUEDWESTEN,NORDWEST,
NORDWESTEN
```

Bei genauer Betrachtung dieser Unterprogramme kann man feststellen, daß sie im Prinzip alle das gleiche tun – sie ändern die Werte von X% und Y%. Wenn wir auch noch diese Information mit in unsere DATA-Anweisungen übernehmen könnten, dann hätten wir ein weiteres Problem gelöst! Wir brauchen nur noch zwei weitere Tabellen anzulegen, um die X- und Y-Koordinaten abzuspeichern, fügen die entsprechenden Werte bei den DATA-Anweisungen hinter der entsprechenden Antwort ein und lesen diese Gesamtinformation jeweils in Viererblöcken (Eingabe, Antwort, X-Bewegung, Y-Bewegung – siehe **Tabelle 2.2**).

```
30 DIM C$(WD%),R$(WD%),X(WD%),Y(WD%)
10000 DATA NORD,NORDEN,0,-1,SUED,SUEDEN,
0,1,WEST,WESTEN,-1,0,OST,OSTEN,1,0
10010 DATA NORDOST,NORDOSTEN,1,-1,SUEDOS
T,SUEDOSTEN,1,1
10020 DATA SUEDWEST,SUEDWESTEN,-1,1,NORD
WEST,NORDWESTEN,-1,-1
11010 READ C$(N),R$(N),X(N),Y(N)
```

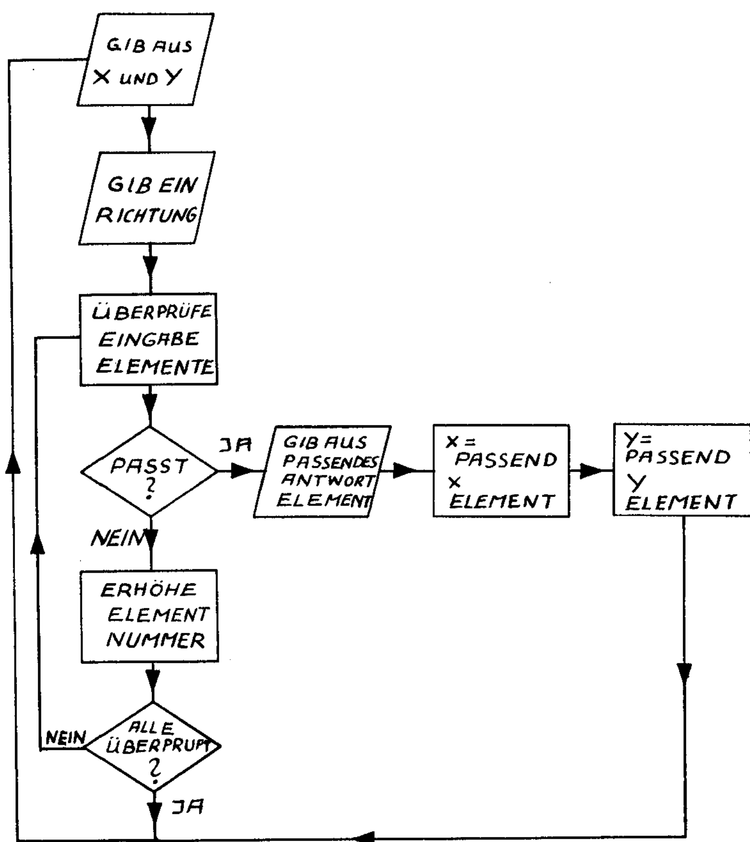
ELEMENT-NUMMER	EINGABE WORT C$(N)	ANTWORT R$(N)	BEWEGUNGSWERT X	– Y
1	NORD	NACH NORDEN	0	–1
1	SUED	NACH SUEDEN	0	1
1	WEST	NACH WESTEN	–1	0
1	OST	NACH OSTEN	1	0
1	NORDOST	NACH NORDOSTEN	1	–1
1	SUEDOST	NACH SUEDOSTEN	1	1
1	SUEDWEST	NACH SUEDWESTEN	–1	1
1	NORDWEST	NACH NORDWESTEN	–1	–1

Tabelle 2.2 In Tabellen eingebaute X- und Y-Verschiebungen

Jetzt können wir die Zeilen 300 bis 2700 löschen und die Zeile 210 so modifizieren, daß X% und Y% hier geändert werden (siehe **Ablaufdiagramm 2.5**).

```
210 IF IN$=C$(N) THEN PRINT G$;R$(N):X%=
X%+X(N):Y%=Y%+Y(N):GOTO 100
```

Diese Vorgehensweise, bei der die gesamte benötigte Information in eine Gruppe von Tabellen abgelegt wird, ist eine weitverbreitete Vorgehensweise und wird noch in vielen weiteren Programmen dieses Buches benutzt.



Ablaufdiagramm 2.5 Nutzung verbundener Felder

WIR KÜRZEN KOMMANDOS

Bis jetzt haben wir immer komplette Wörter als Kommandos benutzt, aber dafür sind viele Fingerübungen notwendig, bis der Computer alle Anweisungen kennt. Da routinierte Programmierer jedoch von Natur aus faul sind, sollte man darüber nachdenken, ob es nicht ausreicht, von jedem Wort nur den ersten Buchstaben zu

benutzen, und deshalb auch nur einen einzelnen Buchstaben einzugeben. In diesem Fall ist darauf zu achten, daß jetzt nur noch Wörter mit unterschiedlichen Anfangsbuchstaben benutzt werden können. Zum Kennzeichnen der bis jetzt benutzten acht Kompaßrichtungen brauchen wir deshalb bis zu zwei Buchstaben, um sie einwandfrei unterscheiden zu können.

```
10000 DATA N,NORDEN,0,-1,S,SUEDEN,0,1,W,
WESTEN,-1,0,O,OSTEN,1,0
10010 DATA NO,NORDOSTEN,1,-1,SO,SUEDOSTE
N,1,1
10020 DATA SW,SUEDWESTEN,-1,1,NW,NORDWES
TEN,-1,-1
```

Es ist Ihnen sicherlich aufgefallen, daß wir wirklich nur die Eingabe abgekürzt haben, und der Computer in der zweiten Tabelle immer noch den vollen Text für die Antwort abgespeichert hat.

WIR PRÜFEN BUCHSTABENWEISE

Bei allen bis jetzt entwickelten Programmen haben wir die exakte Übereinstimmung des Eingabewortes mit dem entsprechenden Schlüsselwort der Tabelle überprüft. Es wäre aber sehr nützlich, wenn wir eine Anzahl von ähnlichen Wörtern ebenfalls als gültige Antwort zulassen würden; z. B. könnte es ausreichen, wenn nur der erste Buchstabe des Eingabewortes mit dem abgekürzten Schlüsselwort übereinstimmt.

```
190 IN$=LEFT$(IN$,1)
```

Dies würde wunderbar mit Norden, Süden, Osten und Westen funktionieren. Bei den Zwischenrichtungen hätten wir jedoch schon Unterscheidungsprobleme. Außerdem gibt es noch eine Fülle von Wörtern, die mit dem Buchstaben N, S, O und W beginnen und nach dieser einfachen Logik als gültige Antwort akzeptiert werden müßten.
Zum Beispiel:

NICHT NORDEN

würde ergeben:

NACH NORDEN

Eine weit sicherere Auswahl kann durch die Benutzung von mehreren Buchstaben getroffen werden. Für unsere vier Haupthimmelsrichtungen würden die ersten drei Buchstaben für eine eindeutige Identifizierung sicher ausreichen. In diesem Fall würden z. B. die Wörter "WES", "WEST" und "WESTLICH" alle akzeptiert werden, und die Wörter "WAS", "WIND", "WUNDE" und "WALD" würden alle zurückgewiesen werden.
Wir brauchen also nur noch die ersten drei Buchstaben der Eingabe LEFT$(IN$,3) mit einer veränderten DATA-Liste zu vergleichen. Die Zeilen 10010 und 10020 werden nicht mehr benötigt, und die Wortanzahlvariable W$% muß den entsprechenden Wert zugewiesen bekommen. Da wir mit dem Element 0 in der Tabelle zu zählen beginnen, muß WD$ in diesem Fall auf den Wert 3 gesetzt werden.

```
20 WD%=3
190 IN$=LEFT$(IN$,3)
10000 DATA NOR,NORDEN,0,-1,SUE,SUEDEN,0,
1,WES,WESTEN,-1,0,OST,OSTEN,1,0
```

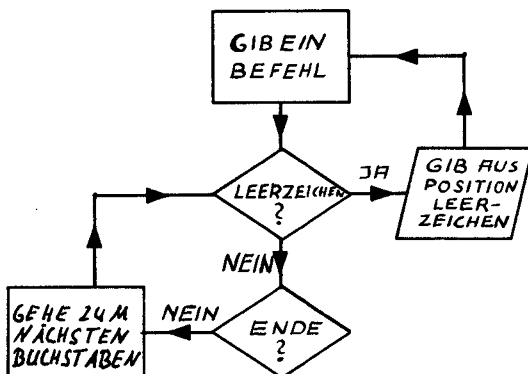
WIR VERWENDEN FOLGEKOMMANDOS

In den bisherigen Programmroutinen haben wir für jede Himmelsrichtung eine diskrete Eingabe bzw. Abfrage durchgeführt. Die Kombination von verschiedenen Richtungen in einer Eingabe würde es uns erlauben, uns auch ohne diese speziellen Eingaben zu jedem Punkt unseres Systems zu bewegen. Wir könnten dann z. B. das Kommando "NORDWEST" als eine Kombination der beiden Kommandos "NORD" und "WEST" behandeln.
Dies konfrontiert uns zum ersten Mal mit der Notwendigkeit, eine Eingabe in ihre einzelnen Wörter zu zerlegen. Stellen Sie sich bitte selbst die Frage, wie Sie eine Folge von Buchstaben als zu einem Wort gehörig erkennen. Richtig! Zwischen zwei Wörtern muß ein Leerzeichen sein. Wir brauchen also nur in unserer Eingabe nach Leerzeichen zu suchen, um die einzelnen "Wörter" zu erkennen.
Die INSTR-Anweisung erlaubt es uns, eine Zeichenkette auf Übereinstimmung mit einer anderen Zeichenkette zu untersuchen. Unglücklicherweise ist diese BASIC-Anweisung nicht im COMMODORE-BASIC enthalten, so daß wir uns dafür eine eigene Routine schreiben müssen. Wir legen sie als Unterprogramm bei der Zeile 5000 ab und benutzen sie für den Rest des Buches als unsere eigenen INSTR-Routine.

```
5000 FOR N=1 TO LEN(IN$)
5010 IF MID$(IN$,N,1)=" "THEN SP%=N :RET
URN
```

```
5020 NEXT N
5030 SP%=0
5040 RETURN
```

Dieses Unterprogramm überprüft, ob das erste Zeichen in IN$ ein Leerzeichen ist. Wenn dies nicht zutrifft, überprüft es automatisch Zeichen für Zeichen, bis das Ende von IN$ erreicht ist. Wenn ein Leerzeichen gefunden wird, gibt der Wert von SP% die Position des Leerzeichens in IN$ an; ist kein Leerzeichen vorhanden, wird SP% auf Null gesetzt (siehe **Ablaufdiagramm 2.6**).



Ablaufdiagramm 2.6 Herausfinden der Position eines Leerzeichens

Zum Aufrufen dieses Unterprogramms vom Hauptprogramm, und zum Ausdrucken des Ergebnisses nach der Rückkehr ins Hauptprogramm brauchen wir folgende Programmzeilen:

```
130 GOSUB 5000
140 PRINT SP% :GOTO 100
```

Versuchen Sie es mit:

NOR WES

4

NORD WEST

5

OR NOR WEST

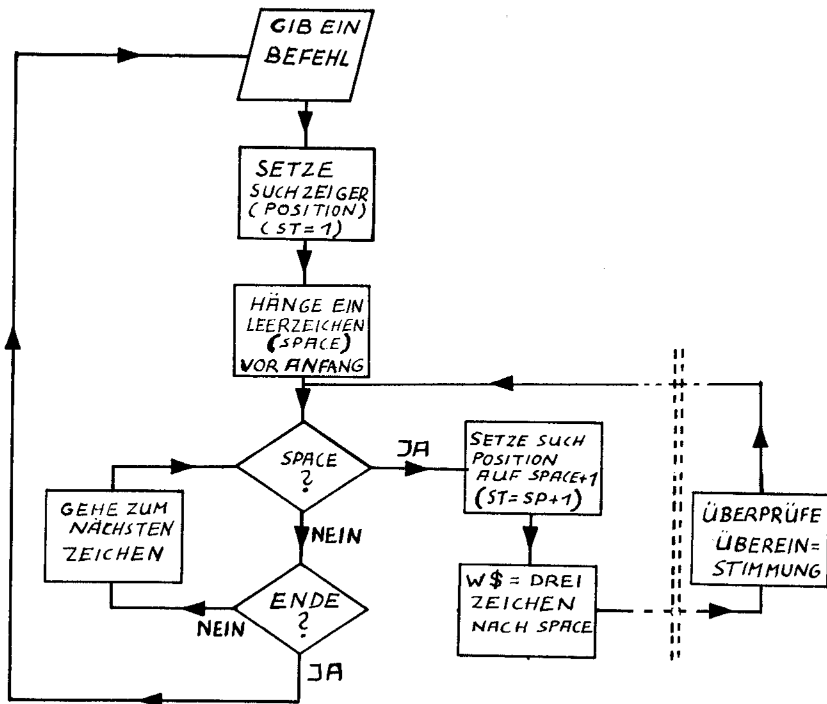
3

Beachten Sie bitte, daß der Wert von SP% immer um 1 höher ist, als die Länge des ersten Wortes, da der Suchprozeß nach dem Auffinden des ersten Leerzeichens abgebrochen wird. Um alle Leerzeichen aufzufinden, müssen wir noch etwas mehr Gedankengut investieren. Aber zuerst löschen wir die Zeile 140, die wir nur für diesen Zwischenschritt benötigt haben.
Lassen Sie uns die Eingabe noch einmal logisch vom Anfang (das heißt, von links) an betrachten. Dazu ersetzen wir LEFT$(IN$,3) durch MID$(IN$,ST%,3), wodurch wir uns jede Drei-Buchstaben-Kombination von IN$ ansehen können. Das Ergebnis dieser Funktion nennen wir W$. Zu Beginn setzen wir die Suchstartposition ST% auf 1 und setzen ein Leerzeichen vor IN$, um auch das erste Wort zu finden (siehe **Ablaufdiagramm 2.7**).

```
125 ST%=1:IN$=" 125 ST%=1:IN$=" "+IN$
130 GOSUB 5000
190 W$=MID$(IN$,ST%,3)
210 IF W$=C$(N) THEN PRINT G$;R$(N):X%=X
%+X(N):Y%=Y%+Y(N):GOTO 100
5000 FOR N=ST% TO LEN(IN$)
```

Wenn dieses Programm, wie oben angegeben, ausgeführt wird, wird nur das erste Wort gefunden, da wir am Ende der Zeile 210 ein GOTO 100 haben. Aber auch das Zurückspringen des Programms zu unserer INSTR-Routine in Zeile 130 löst das Problem nicht, da es die Überprüfung immer wieder mit dem Anfang von IN$ beginnen wird und jedesmal dasselbe Leerzeichen findet. Wenn wir erst einmal das erste Leerzeichen gefunden haben, müssen wir zur Überprüfung des nächsten Zeichens die Startposition ST% um 1 erhöhen. Wenn keine weiteren Leerzeichen gefunden werden können, haben wir das Ende der Eingabe erreicht und können wieder zur Zeile 100 springen.

```
140 IF SP%>0 THEN ST%=SP%+1:GOTO 190
150 GOTO 100
210 IF W$=C$(N) THEN PRINT G$;R$(N):X%=X
%+X(N):Y%=Y%+Y(N):GOTO 130
```



Ablaufdiagramm 2.7 Suche nach einem Schlüsselwort

Die Eingabe:

NORD WEST

ergibt:

NACH NORDEN
NACH WESTEN

und sogar:

NOR NOR OST

wird erkannt als:

NACH NORDEN
NACH NORDEN
NACH OSTEN

Es wäre sehr viel sinnvoller, wenn wir all diese unnötigen "nach" nicht mehr ausgeben würden und alle Richtungen hintereinander in derselben Zeile ausdrukken könnten. Wir brauchten dann G$ nur einmal direkt vor der INSTR-Routine auszudrucken. Jedesmal, wenn wir durch die Scheife gehen, die das eingegebene Wort mit den abgespeicherten Schlüsselworten vergleicht, drucken wir R$(N); aber nur, wenn wir wirklich eine Übereinstimmung gefunden haben. Da wir hinter dem Druckbefehl ein Semikolon angegeben haben, wird kein Zeilenvorschub generiert, und die Wörter werden in der gleichen Zeile ausgedruckt. Damit sie gut lesbar sind, trennen wir sie durch ein Leerzeichen. Um nach Abschluß der Untersuchung einer Eingabe die entsprechenden Ergebnisse in der nächsten Zeile drucken zu können, fügen wir in Zeile 145 ein einfaches PRINT ein.

```
126 PRINT G$;
145 PRINT
210 IF W$=C$(N) THEN PRINT R$(N);" ";:X%
=X%+X(N):Y%=Y%+Y(N):GOTO 130
```

Die Eingabe:

NORDEN OSTWAERTS SUEDLICH WEST

schickt Sie im Kreis zum Ausgangspunkt zurück:

NORDEN OSTEN SUEDEN WESTEN

KAPITEL 3

VERSTEHEN DER UMGANGSSPRACHE

Bis jetzt konnten wir uns mit dem Computer nur in einer sehr beschränkten Art und Weise unterhalten, da wir ihn nur programmiert hatten, einige Wörter oder Buchstaben zu verstehen, und er diese nur erkannt hat, wenn wir sie genau in der vorgegebenen Art eingegeben haben. Wenn Sie z. B. bei der Eingabe ein Leerzeichen vor oder hinter Ihrem Kommando eingefügt haben, wird es zurückgewiesen. Dies geschieht, weil wir die beiden Zeichenketten auf genaue Übereinstimmung überprüfen.

In der Wirklichkeit benutzen wir die sogenannte "Umgangssprache", die eine ziemlich komplizierte und außerordentlich variable Angelegenheit ist, mit der normalerweise nur ein menschliches Gehirn umgehen kann. Sogar wenn wir Unterschiede, wie zwischen "Deutsch" und "Schweizerdeutsch" oder landschaftliche Dialekte außer acht lassen, ergeben sich beim Arbeiten mit der Umgangssprache eine unendliche Anzahl von Problemen.

Sogar das höchstentwickelte System der Welt kann nicht mit allen sprachlichen Problemen fertig werden. Es gibt eine alte Geschichte, die Ihnen dieses Problem wunderbar illustriert. Der CIA entwickelte ein exzellentes Übersetzungsprogramm, das in der Lage war, sofort von englisch nach russisch und wieder zurück zu übersetzen.

In der Hoffnung, den Präsidenten der Vereinigten Staaten damit beeindrucken zu können, demonstrierten sie ihm die Möglichkeiten dieses Programms. Das Programm übersetzte alles, was der Präsident sagte, sofort in russisch, wiederholte es und übersetzte es dann wieder zurück in die englische Sprache. Er war außerordentlich beeindruckt und beschäftigte sich nur noch mit dem System, bis ihn einer seiner Mitarbeiter daran erinnerte, daß seine Frau draußen auf ihn wartete. Als er zu seiner Entschuldigung bemerkte: "Aus den Augen, aus dem Sinn", war er höchst erstaunt, daß ihm die Maschine mit "Unsichtbarer Verrückter!" antwortete.

BEHANDLUNG VON SÄTZEN

Jeder weiß, daß die wirkliche Sprache aus Sätzen besteht; aber was verstehen wir wirklich genau unter einem Satz? Nun gut, die offensichtlichste Art, einen Satz zu erkennen, ist, wenn wir einen Punkt sehen. Aber wenn wir uns wirklich mit Sätzen beschäftigen wollen, müssen wir schon ein bißchen mehr darüber wissen.

Ein Satz wird im allgemeinen definiert als "eine Folge von gesprochenen oder geschriebenen Wörtern, die eine grammatikalische vollständige Aussage bildet und

normalerweise ein Subjekt und Prädikat enthält, mit der man Aussagen, Fragen, Aufforderungen oder Anforderungen" ausdrücken kann oder auch die simple Definition "Teil der Sprache oder eines Schreibens zwischen zwei Satzpunkten". So, da haben wir unser Problem zurück. Kann das jemand in normales Deutsch übersetzen? Wie kann man von einem Computer erwarten, daß er mit all den Schwierigkeiten und unlogischen Regeln der deutschen Sprache zurecht kommt? In Ordnung, lassen Sie uns zuerst ein paar ganz einfache Satzbeispiele betrachten.

ICH WILL.

Besteht aus dem Subjekt ICH und dem Verb WILL.

ICH WILL BISQUIT.

hat auch noch das Objekt BISQUIT.

ICH WILL SCHOKOLADEN BISQUIT.

erweitert das Objekt mit dem Adjektiv SCHOKOLADEN.

ICH WILL MANCHMAL SCHOKOLADEN BISQUIT.

erweitert das Verb mit dem Adverb MANCHMAL.

Das wichtigste Wort in allen Beispielen war "will", da es die wichigste Idee dieser Sätze zum Ausdruck bringt. Das zweite Beispiel war schon informativer als das erste, da es herausstellte, daß Sie nur ein ganz bestimmtes Objekt, nämlich die BISQUIT, möchten. Durch das Hinzufügen des Adjektivs SCHOKOLADEN haben wir weitere Informationen über die Art des gewünschten Objekts, aber durch das Hinzufügen des Adverbs MANCHMAL wurden wir doch wieder ziemlich verunsichert.

Wie kann nun ein Computer-Programm solche Sätze entschlüsseln? Die Antwort kann nur in einer logischen Struktur dieser Sätze liegen. Welche "Regeln" können wir für diese Beispiele finden?

1. Alle Beispiele begannen mit dem Subjekt ICH und endeten mit einem Punkt.

2. Das letzte Wort war immer das Objekt BISQUIT (es sei denn, es gab kein Objekt, und der Satz bestand nur aus zwei Wörtern).

3. Wenn das Wort vor dem Objekt nicht das Verb WILL war, war es das Adjektiv SCHOKOLADEN oder das Adverb MANCHMAL.

Lassen Sie uns jetzt ein Programm schreiben, in dem wir dem Computer Sätze übergeben, die er in ihre Teile auflösen soll.
Zu Beginn geben wir ihm ein Vokabular von Objekten, Adjektiven und Adverbien, mit denen er arbeiten kann. Wir lesen (READ) diese von den DATA-Anweisungen und legen sie dann in den Tabellen OB, AJ und AV ihres Typs entsprechend ab.

```
10 GOSUB 10000
10000 DIM OB$(5),AJ$(5),AV$(2)
10010 DIM W$(4)
10999 REM OBJEKTE
11000 DATA BISQUIT,ZWIEBACK,KUCHEN
11010 DATA KAFFEE,TEE,WASSER
11019 REM ADJEKTIVE
11020 DATA SCHOKOLADEN,INGWER,MARMELADEN

11030 DATA KALTEN,HEISSE,LAUWARM
11039 REM ADVERBIEN
11040 DATA IMMER,OFT,MANCHMAL
11100 FOR N=0 TO 5
11110 READ OB$(N)
11120 NEXT N
11130 FOR N=0 TO 5
11140 READ AJ$(N)
11150 NEXT N
11160 FOR N=0 TO 2
11170 READ AV$(N)
11180 NEXT N
11190 RETURN
```

Als nächstes müssen wir die Sätze in Wörter aufteilen (siehe **Ablaufdiagramm 3.1**). Das machen wir wieder, indem wir mit unserer INSTR-Routine nach Leerzeichen suchen. Zur Erleichterung fügen wir am Ende von IN$ ein Leerzeichen an, so daß das Format des letzten Wortes des Satzes wie das aller anderen ist.

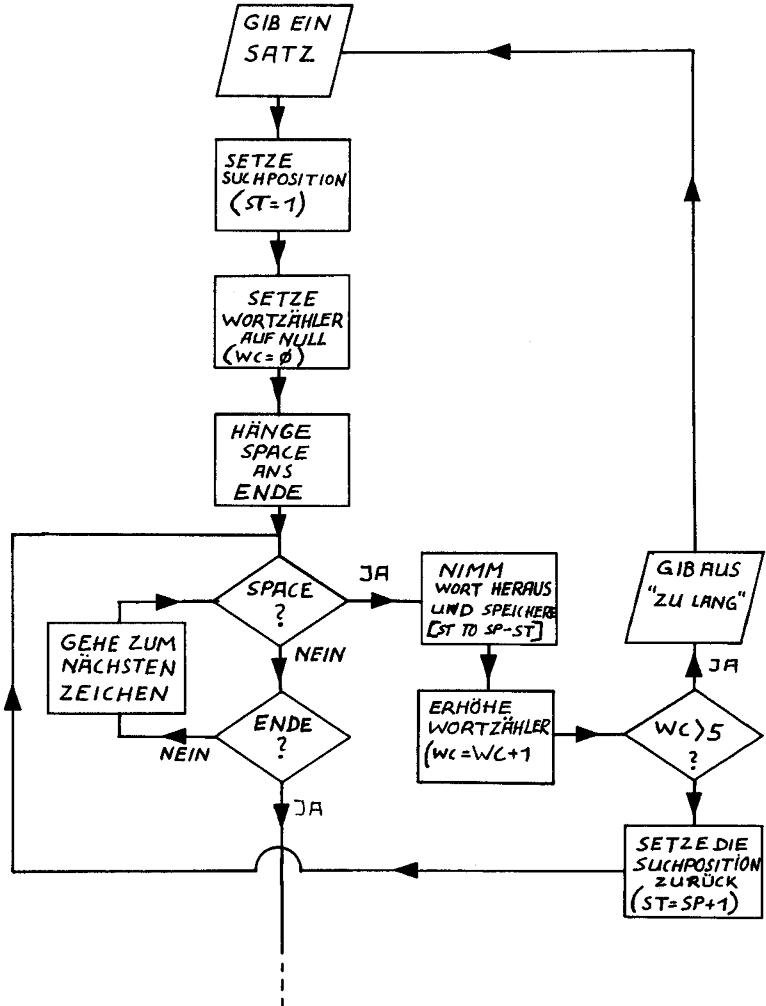
```
100 INPUT IN$
120 IN$=IN$+" "
130 GOSUB 5000
190 GOTO 130
```

Das Ende des Satzes ist erreicht, wenn keine Leerzeichen mehr gefunden werden.

```
140 IF SP%=0 THEN 200
```

Wenn ein Leerzeichen gefunden wird, wird der Bereich von IN$, beginnend bei ST% (laufende Suchposition) bis SP%-ST% (laufendes Leerzeichen minus laufende Suchposition = Länge des Wortes), herausgelöst und in einer Wortspeichertabelle W$(WC%) abgelegt.

```
150 W$(WC%)=MID$(IN$,ST%,SP%-ST%)
10010 DIM W$(4)
```



Ablaufdiagramm 3.1 Worte herausfinden

Wir beginnen mit ST%=1, so daß die Suche wirklich mit dem ersten Zeichen der Eingabe beginnt.

Der Wortzähler WC% wird auf Null gesetzt, so daß das erste gefundene Wort im Element 0 der Wortspeichertabelle W$ abgelegt wird.

```
110 ST%=1:WC%=0
```

Der Wortzähler wird erhöht (so daß das nächste Mal das nächste Element der Tabelle W$ benutzt wird), und der Satz wird darauf überprüft, ob er nicht mehr als fünf Wörter enthält. Der Start für die nächste Überprüfung wird dann auf die Position hinter dem letzten Leerzeichen gesetzt, und die Überprüfung wird fortgesetzt.

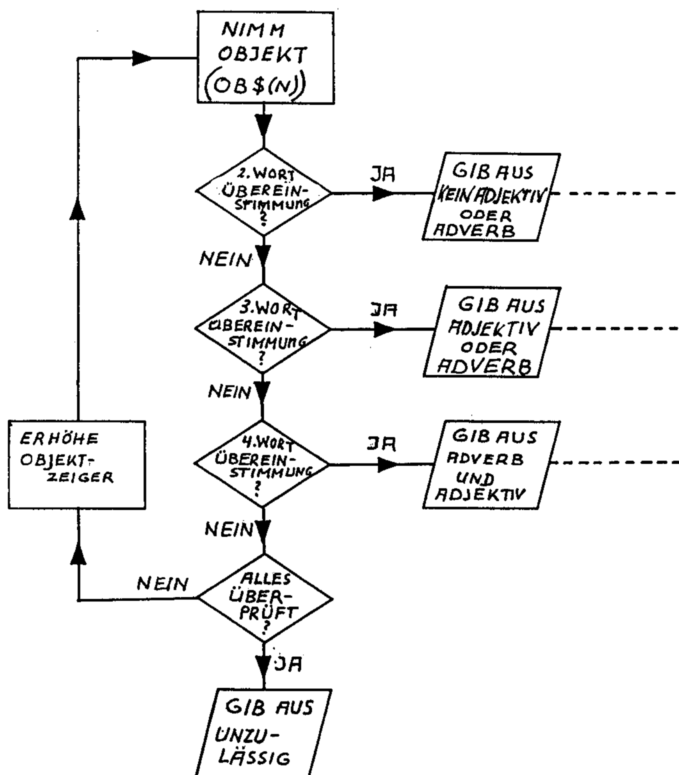
```
160 WC%=WC%+1
170 IF WC%>5 THEN PRINT "SATZ ZU LANG ":
GOTO 100
180 ST%=SP%+1
```

Nun wird überprüft, ob wir eine Übereinstimmung zwischen den Schlüsselwörtern des Satzes und den in der Tabelle OB$(N) abgelegten Objekten finden (siehe **Ablaufdiagramm 3.2**). Dabei werden nur die Wörter zwei bis vier (das heißt, das dritte, vierte oder fünfte Wort des Satzes) überprüft. Bei unserem sehr einfachen Beispiel kann das Objekt nur in diesen Positionen auftreten. Entsprechend der gefundenen Position springt das Programm zu drei verschiedenen Routinen. Sollte überhaupt keine Übereinstimmung gefunden werden, wird eine entsprechende Meldung ausgegeben und eine neue Eingabe angefordert.

```
190 GOTO130
200 FOR N=0 TO 5
210 IF W$(2)=OB$(N) THEN 500
220 IF W$(3)=OB$(N) THEN 600
230 IF W$(4)=OB$(N) THEN 700
240 NEXT N
250 PRINT "OBJEKT NICHT GEFUNDEN"
260 GOTO 100
```

Wenn das Objekt als Wort 2 gefunden wurde, gab es in diesem Satz weder ein Adjektiv noch ein Adverb.

```
500 PRINT "KEIN ADJEKTIV ODER ADVERB"
510 GOTO100
```



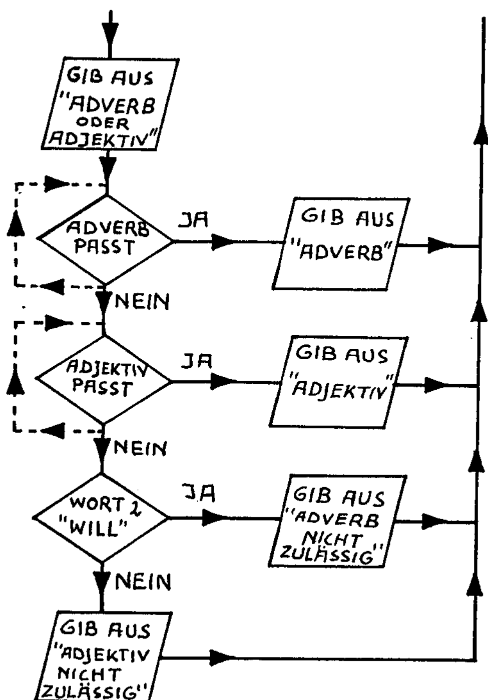
Ablaufdiagramm 3.2 Suche nach einer Übereinstimmung

Wenn das Objekt als Wort 3 gefunden wurde, kann der Satz entweder ein Adjektiv oder ein Adverb enthalten (siehe **Ablaufdiagramm 3.3**).

```
600 PRINT"ENTWEDER ADJEKTIV ODER ADVERB"
```

Zuerst überprüfen wir, ob das Wort 2 mit irgendeinem Inhalt der Adverbtabelle übereinstimmt.

```
610 FOR N=0 TO 2
620 IF W$(2)=AV$(N) THEN 300
630 NEXT N
```



Ablaufdiagramm 3.3 Adverb oder Adjektiv

Wenn keine Übereinstimmung gefunden wird, müssen wir das Wort 2 noch gegen die Adjektivtabelle überprüfen.

```
640 FOR N=0 TO 5
650 IF W$(2)=AJ$(N) THEN 1000
660 NEXT N
```

Wenn in einem dieser beiden Tests das Eingabewort in der entsprechenden Tabelle gefunden wurde, wird eine entsprechende Meldung ausgegeben. Beachten Sie jedoch, daß diese beiden Möglichkeiten exklusiv sind, das heißt, daß in einem Satz mit vier Wörtern entweder nur ein Adverb oder ein Adjektiv möglich ist.

```
670 IF W$(1)<>"WILL" THEN PRINT "ADVERB"
;W$(1);" UNVERSTAENDLICH ":GOTO100
680 PRINT "ADJEKTIV ";W$(2);" UNVERSTAEN
DLICH ":GOTO100
```

```
900 PRINT "ADVERB"
910 GOTO 100
1000 PRINT "ADJEKTIV"
1010 GOTO 100
```

Wenn ein Adverb und ein Adjektiv vorhanden sind, müssen wir den Satz auf beide Möglichkeiten überprüfen; das heißt, nach dem erfolgreichen ersten Test muß auch der zweite Test durchgeführt werden (siehe **Ablaufdiagramm 3.4**).

```
700 PRINT "ADVERB UND ADJEKTIV"
710 FOR N=0 TO 2
720 IF W$(2)=AV$(N) THEN 750
730 NEXT N
```

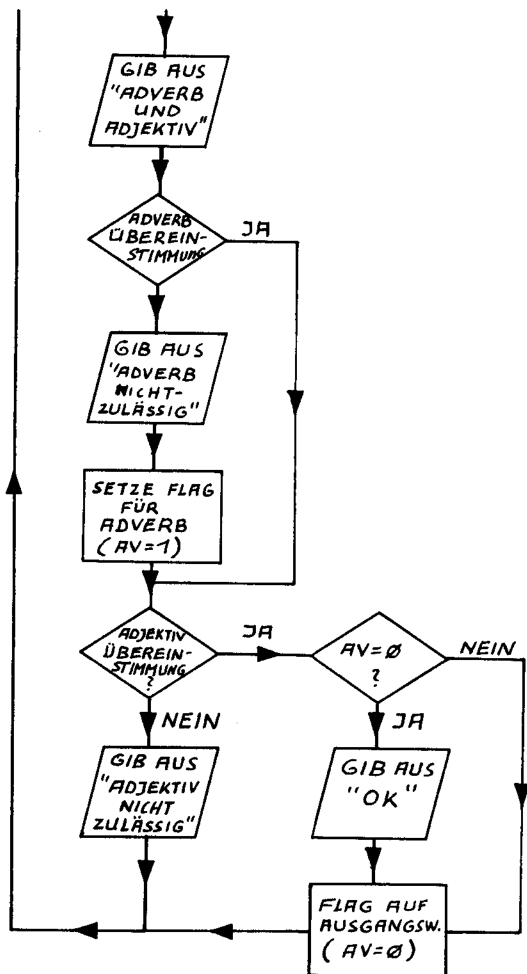
Wenn kein passendes Adjektiv gefunden wird, wird eine entsprechende Meldung ausgegeben und das Kennzeichen AV% auf 1 gesetzt, bevor auf Adjektiv geprüft wird.

```
740 PRINT "ADVERB ";W$(2);" UNVERSTAENDL
ICH ":AV%=1
750 FOR N=0 TO 5
760 IF W$(3)=AJ$(N) THEN 800
770 NEXT N
```

Wenn kein passendes Adjektiv gefunden werden kann, springt das Programm nach entsprechender Fehlerausgabe wieder an den Anfangspunkt zurück.

```
780 PRINT "ADJEKTIV ";W$(3);" UNVERSTAEN
DLICH "
790 GOTO 100
```

Wenn das Adjektiv jedoch gefunden wurde, wird zusätzlich überprüft, ob das Fehlerkennzeichen nicht gesetzt war. In diesem Fall wird eine positive Meldung ausgegeben; in beiden Fällen wird das Fehlerkennzeichen auf Null gesetzt, bevor die nächste Eingabe angefordert wird.

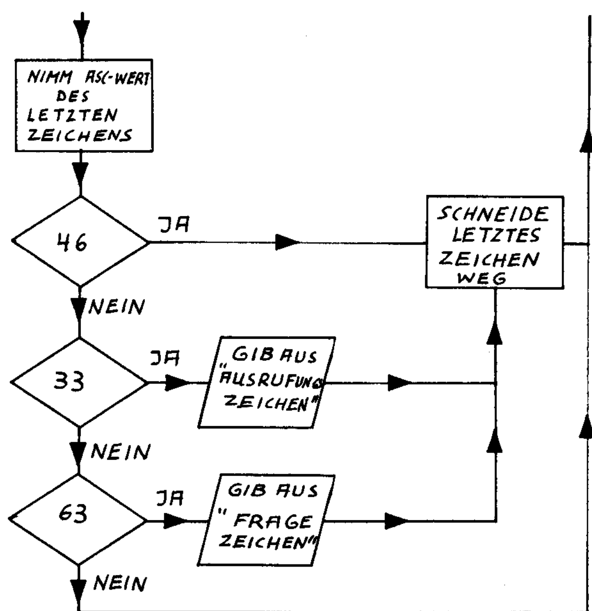


Ablaufdiagramm 3.4 Adverb und Adjektiv

```
800 IF AV%=0 THEN PRINT "ADJEKTIV UND AD
VERB OK"
810 AV%=0
820 GOTO 100
```

WIR BERÜCKSICHTIGEN DIE INTERPUNKTION

Wie wir bereits wissen, wird normalerweise das Ende eines Satzes durch den Punkt erkannt. Was passiert aber, wenn ein besonders "kluger" Benutzer anstelle des Punktes das Ausrufezeichen oder das Fragezeichen benutzt? Sie können sich vorstellen, daß das erhebliche Probleme geben kann, da das letzte Wort jetzt um das Fragezeichen oder das Ausrufezeichen verlängert ist.
Wir müssen deshalb das letzte Zeichen unserer Eingabe IN$ auf den Punkt überprüfen; das einfachste ist, es auf den ASCII-Code zu überprüfen. Ist die Prüfung positiv, dann "ignorieren" wir dieses Zeichen und fahren wie gewohnt fort.



Ablaufdiagramm 3.5 Behandlung von Satzzeichen

Wir bringen diese Abprüfungen in einem Unterprogramm unter, das wir sofort nach dem Einlesen aufrufen. Da wir auch noch die Prüfungen auf das Fragezeichen und das Ausrufezeichen durchführen müssen, speichern wir den ASCII-Code des letzten Eingabezeichens in der Variablen LC% ab (siehe **Ablaufdiagramm 3.5**).

```
105 PRINT:GOSUB 2000
```

```
2000 LC%=ASC(RIGHT$(IN$,1))
2010 IF LC%=46 THEN PRINT"SATZENDE":GOTO
2100
2090 RETURN
2100 IN$=LEFT$(IN$,LEN(IN$)-1):RETURN
```

Wir können die Ausrufezeichen oder Fragezeichen ebenfalls durch den Vergleich mit ihrem ASCII-Code herausfinden und ihre Anwesenheit durch eine entsprechende Ausgabe kennzeichnen.

```
2020 IF LC%=33 THEN PRINT"FESTSTELLUNG":
GOTO2100
2030 IF LC%=63 THEN PRINT"FRAGESATZ":GOT
O2100
```

Die normale INPUT-Anweisung erlaubt keine weiteren Eingaben hinter einem Komma, da dieses als Schlußzeichen erkannt wird. Aber wir können uns unter Verwendung von GET eine Routine schreiben, die jeden Test einschließlich Kommas akzeptiert. Als erstes "leeren" wir den Eingabebereich IN$ und geben ein '<' als Cursor aus.

```
100 IN$="":PRINT" <";
```

In der Zeile 101 wird auf einen Tastendruck abgeprüft. Wenn keine Taste gedrückt ist, wird diese Überprüfung laufend wiederholt.

```
101 GET I$:IF I$="" THEN 101
```

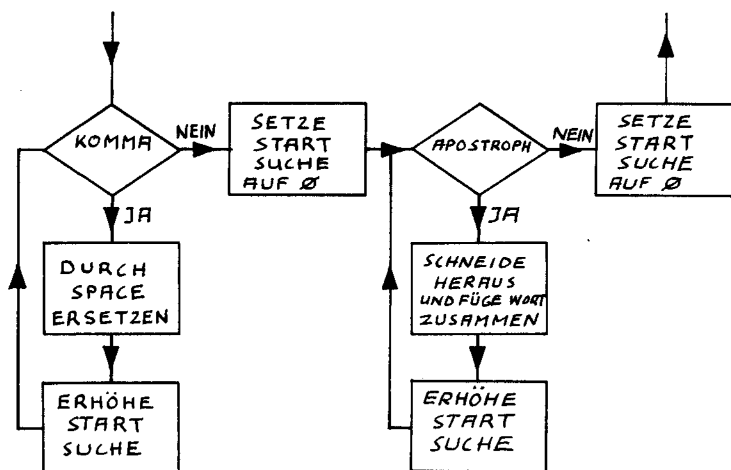
Wenn eine Taste gedrückt ist, wird ein "Cursor links"-Symbol gefolgt von dem Zeichen der gedrückten Taste ausgegeben. Dieses Zeichen wird dann an den Eingabebereich IN$ angefügt, und das Programm springt zurück zu der Prüfroutine in Zeile 101. Dadurch erscheint jede Eingabe wir bei einem normalen INPUT auf dem Bildschirm, und jeder Fehler kann mit der "Backspace"-Taste korrigiert werden.

```
103 PRINT"█";I$;"<";:IN$=IN$+I$:GOTO 101
```

Das Ende einer Eingabe kann durch das Abprüfen der RETURN-Taste erkannt werden, die den ASCII-Code 13 hat. Wenn die Eingabe vollständig ist, wird der Cursor auf die nächste Zeile gesetzt.

```
102 IF ASC(I$)=13 THEN PRINT:GOTO 105
```

Kommata können sehr gut benutzt werden, um Sätze in einzelne Teile zu unterteilen, die dann wiederum als "Teilsätze" für sich selbst untersucht werden können. Normalerweise ist es jedoch besser, die Kommata durch Leerzeichen zu ersetzen, bevor der Satz in Wörter unterteilt wird (siehe **Ablaufdiagramm 3.6**).



Ablaufdiagramm 3.6 Ersetzen vom Kommata und Apostrophen

Sie müssen jedoch darauf achten, daß kein Leerzeichen nach dem Komma folgen darf, da jedes Leerzeichen hinter einem ausgetauschten Komma wieder als Anfang eines neuen Wortes erkannt wird.
Anstatt eine vollkommen neue INSTR-Routine zu schreiben, modifizieren wir unsere vorhandene Routine so, daß wir unseren Eingabebereich IN$ auf jede vordefinierte Zeichenkette TA$ abprüfen lassen. Für einen zukünftigen Gebrauch merken wir uns die Position der Übereinstimmung mit Hilfe der Variablen IS%, deren Wert wir dann wieder jeder anderen Variablen, z. B. SP%, zuweisen können. Aber zuerst müssen wir jedoch unsere "Leerzeichenabprüfung" auf das neue Format umstellen.

```
130 TA$=" ":GOSUB 5000:SP%=IS%
5010 IF MID$(IN$,N,LEN(TA$))=TA$ THEN IS
%=N:RETURN
5030 IS%=0
```

Mit der gleichen Methode können wir natürlich auch das Komma suchen, bevor es durch ein Leerzeichen ersetzt wird.

```
115 TA$=",":GOSUB 3000
3000 GOSUB 5000:CM%=IS%
3010 IF CM%=0 THEN ST%=1:RETURN
3020 IN$=LEFT$(IN$,CM%-1)+" "+RIGHT$(IN$
,LEN(IN$)-CM%)
3030 SP%=CM%+1
3040 GOTO3000
```

Wenn Sie die folgende Zeile in das Programm übernehmen, können Sie prüfen, ob die Satzzeichen wirklich aus der Eingabe entfernt worden sind.

```
3025 PRINT IN$
```

Anführungszeichen können in der gleichen Art gefunden werden; sie werden aber nicht durch ein Leerzeichen ersetzt, sondern die Wörter werden zusammengezogen.

```
115 TA$=",":GOSUB 3000:TA$="'":GOSUB3100

3100 GOSUB 5000:AP%=IS%
3110 IF AP%=0 THEN ST%=1:RETURN
3120 IN$=LEFT$(IN$,AP%-1)+RIGHT$(IN$,LEN
(IN$)-AP%)
3125 PRINT IN$
3130 ST%=AP%+1
3140 GOTO 3100
```

DAS GLEITENDE SUCHEN

Wenn wir den ganzen Satz von links nach rechts gleitend auf jedes Schlüsselwort untersuchen, ohne den Satz zuerst in Wörter aufzuteilen, können wir bei der Eingabe ein vollkommen freies Format benutzen. Hierbei nehmen wir das erste

Schlüsselwort und überprüfen den Eingabebereich IN$ in der gleichen Länge auf Übereinstimmung, beginnend mit dem ersten Buchstaben. Bei Nichtübereinstimmung gleiten wir jedesmal einen Buchstaben nach rechts und wiederholen den Test so lange, bis wir entweder eine Übereinstimmung erkannt haben oder das Ende von IN$ erreicht ist. Wenn z. B. IN$ "ICH WILL KUCHEN" und das erste Schlüsselwort "KUCHEN" war, würden die Vergleiche folgendermaßen aussehen:

1. Durchgang	ICH WI
2. Durchgang	CH WIL
3. Durchgang	H WILL
.	
.	
.	
8. Durchgang	L KUCH
9. Durchgang	KUCHE
10. Durchgang	KUCHEN (Übereinstimmung)

Bis jetzt haben wir mit unserer INSTR-Routine nur jeweils ein einzelnes Zeichen gesucht, und wir müssen die Programmzeile 5010 so modifizieren, daß bei der Untersuchung die Länge der Zeichenkette TA$ berücksichtigt wird.

```
5010 IF MID$(IN$,N,LEN(TA$))=TA$ THEN IS
%=N:RETURN
```

Die Zeilen 105–1010 sind zu löschen. Dafür ist die Zeile 210 zum Abprüfen auf das erste Objekt OB$(0) einzufügen.

```
210 TA$=OB$(M):GOSUB 5000:SP%=IS%:IF SP%
>0 THEN PRINT OB$(M);" ";
```

Durch die Benutzung einer Schleife können wir die Vergleiche auf jedes Objekt in der gleichen Art durchführen. Durch das Semikolon hinter dem Druckbefehl von OB$(M) wird sichergestellt, daß jedes Wort auf der gleichen Zeile ausgegeben wird.

```
200 FOR M=0 TO 5
220 NEXT M
```

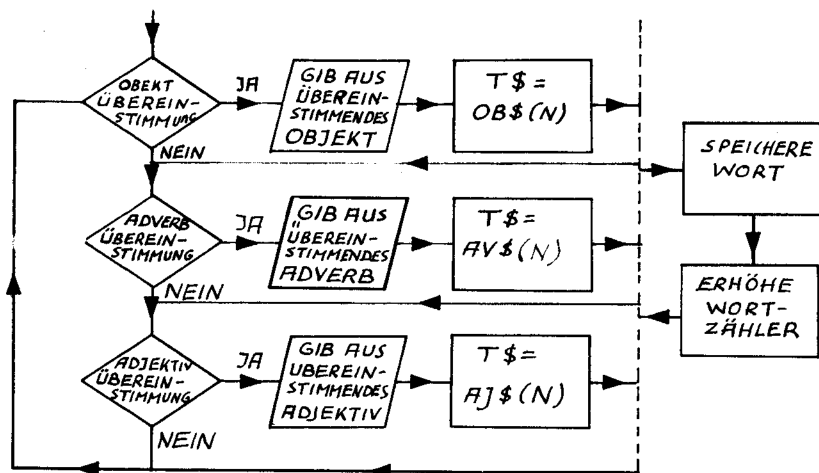
In der gleichen Art und Weise können wir auf die Adverbien und die Adjektive prüfen.

```
300 FOR M=0 TO 2
310 TA$=AV$(M):GOSUB 5000:SP%=IS%:IF SP%
>0 THEN PRINT AV$(M);" ";
320 NEXT M
400 FOR M=0 TO 5
410 TA$=AJ$(M):GOSUB 5000:SP%=IS%:IF SP%
>0 THEN PRINT AJ$(M);" ";
420 NEXT M
1000 GOTO 100
READY.
```

Um auszugeben, was wir gefunden haben, und um die entdeckten Wörter später wieder verwenden zu können, speichern wir jedes sofort in einer Tabelle ab. Unsere Wortspeichertabelle W$ erweitern wir auf eine Kapazität von 20 Wörtern, damit wir auch längere Sätze behandeln können.

```
10010 DIM W$(19)
```

Wenn eine Übereinstimmung erkannt wird, wird das gefundene Wort in der Zeichenkette T$ zwischengespeichert, und ein Unterprogramm in der Zeile 1500 aufgerufen, daß dieses Wort in das erste Element der Wortspeichertabelle ablegt (siehe **Datenflußdiagramm 3.7**).



Ablaufdiagramm 3.7 Ansteigende Suche

```
210 TA$=OB$(M):GOSUB5000:SP%=IS%:IF SP%>
0 THEN T$=OB$(M):PRINT T$;" ";:GOSUB1500
```

```
1500 W$(WC%)=T$
```

Vor dem Rücksprung wird der Wortzähler WC% erhöht, damit beim nächsten Durchlauf das nächste Wort in das nächste Tabellenelement abgelegt werden kann.

```
1520 WC%=WC%+1
1530 RETURN
```

Da wir in dem Unterprogramm den Zwischenspeicher T$ benutzen, können wir ihn bei den Tests auf Adverbien und Adjektive in der gleichen Art benutzen.

```
310 TA$=AV$(M):GOSUB5000:SP%=IS%:IF SP%>
0 THEN T$=AV$(M):PRINT T$;" ";:GOSUB1500
```

```
410 TA$=AJ$(M):GOSUB5000:SP%=IS%:IF SP%>
0 THEN T$=AJ$(M):PRINT T$;" ";:GOSUB1500
```

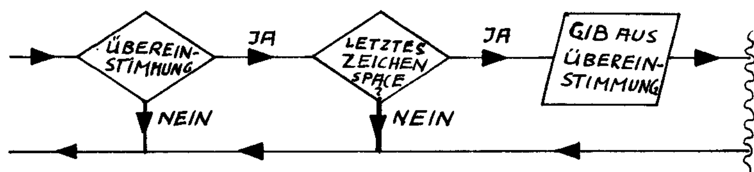
TEILWEISE ÜBEREINSTIMMUNG

Ein besonderer Vorteil der "gleitenden Suchmethode" ist die Möglichkeit, eine Reihe von aufeinander folgenden Wörtern durch die Überprüfung auf einige Schlüsselzeichen zu erkennen. Dadurch ist es nicht mehr nötig, für Hauptwörter Einzahl und Mehrzahl eingeben zu müssen, z. B. BISQUIT und BISQUITS. Wenn Sie die DATA-Zeile 11000 folgendermaßen berichtigen, werden beide anerkannt.

```
11000 DATA BISQUIT,ZWIEBACK,KUCHEN
```

Aber auch diese Vorgehensweise kann uns große Überraschungen präsentieren. Wenn wir z. B. Kaffee und Kaffees finden wollen, werden uns auch KAFFEESATZ, KAFFEESAHNE oder KAFFEETASSE als richtig angegeben.

Dasselbe Problem haben wir natürlich mit vorangesetzten Zeichen, da der Computer z. B. nicht zwischen IMMER und NIMMER unterscheiden kann. Sie sollten deshalb in Ihr Programm eine Prüfung einbauen, ob das Zeichen vor dem Beginn einer Übereinstimmung ein Leerzeichen war, womit sichergestellt ist, daß es sich hier um den Beginn eines Wortes handelt (siehe **Ablaufdiagram 3.8**). Da SP% die



Ablaufdiagramm 3.8 Test ob dies Wortbeginn

laufende Startposition für das Wort angibt, verweist MID$ (IN$,SP%−1,1) auf das unmittelbar davorliegende Zeichen.

```
210 TA$=OB$(M):GOSUB 5000:SP%=IS%:IF SP%
=0 THEN NEXT M:GOTO 300
211 IF MID$(IN$,SP%-1,1)<>" " THEN NEXT
M:GOTO 300
212 T$=OB$(M):PRINT T$;" ";:GOSUB 1500
310 TA$=AV$(M):GOSUB 5000:SP%=IS%:IF SP%
=0 THEN NEXT M:GOTO 400
311 IF MID$(IN$,SP%-1,1)<>" " THEN NEXT
M:GOTO 400
312 T$=AV$(M):PRINT T$;" ";:GOSUB 1500
320 NEXT M
410 TA$=AJ$(M):GOSUB 5000:SP%=IS%:IF SP%
=0 THEN NEXT M:GOTO 440
411 IF MID$(IN$,SP%-1,1)<>" " THEN NEXT
M :GOTO 440
412 T$=AJ$(M):PRINT T$;" ";:GOSUB 1500
```

Damit das Ganze auch beim ersten Wort fehlerfrei funktioniert, müssen Sie ein Leerzeichen vor den Anfang von IN$ setzen.

```
110 IN$=" "+IN$
```

Um die fehlerhaft erkannten Wörter einzuschränken, können Sie Überprüfungen auf den nächsten Buchstaben nach der Übereinstimmung oder auf die Länge des Wortes durchführen.

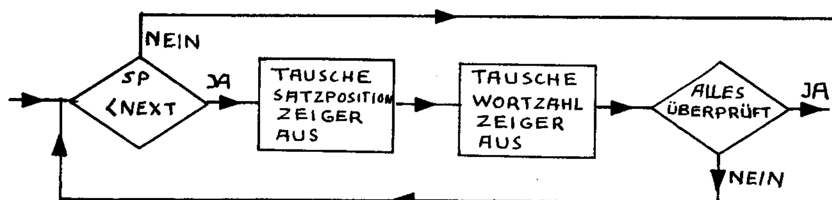
WIR BRINGEN ALLES IN DIE RICHTIGE REIHENFOLGE

Wir haben jetzt zwar alle Wörter in dem Satz gefunden, unabhängig davon, wo sie gestanden haben und welche anderen Zeichen noch vorhanden waren, aber sie sind in der Reihenfolge gespeichert, in der sie in DATA angegeben sind. Der Grund dafür ist, daß wir den Vergleich mit dem ersten Element der Objekttabelle begonnen haben und nicht mit dem ersten Wort im Satz. Es wäre zu begrüßen, wenn wir die Wortspeichertabelle so umarrangieren könnten, daß die darin gespeicherten Wörter in der gleichen Reihenfolge wie in dem Originalsatz abgelegt sind.
Dazu müssen wir uns jeweils die Position des Wortes im Satz SP% und den Wortzähler WC% merken, da die Übereinstimmung für jedes Wort in einer neuen Wortpositionstabelle WP% abgeprüft wird. Wir tun dies mit einer zweidimensionalen Tabelle, in der wir die Satzposition im ersten Element WP (WC%,0) und den Wortzähler WP (WC%,1) im zweiten Element ablegen.

```
10020 DIM WP(19,1)
1510 WP(WC%,0)=SP%:WP(WC%,1)=WC%
```

Das Unterprogramm zum Neusortieren beginnt bei Zeile 4000 und muß immer dann aufgerufen werden, wenn eine Übereinstimmung gefunden wurde.

```
440 IF WC%=0 THEN 470
450 GOSUB 4000
460 GOTO 100
470 PRINT"KEINE UEBEREINSTIMMUNG GEFUNDE
N"
480 GOTO 100
```



Ablaufdiagramm 3.9 Die Worte in richtiger Reihenfolge

Das Sortierprogramm (siehe **Ablaufdiagramm 3.9**) nimmt die Satzposition des ersten gefundenen Wortes (erstes Element in der ersten Dimenison WP(0,0)) und vergleicht es mit der Satzposition des zweiten gefundenen Wortes (zweites Element in der ersten Dimension WP(0+1,0)). Wenn die Position des ersten Wortes größer ist als die des zweiten, dann ist das erste Wort im Satz rechts vom zweiten Wort, und beide müssen vertauscht werden. Dies korrigiert die Satzpositionszähler, aber die Wortzähler müssen auch noch durch Vertauschen korrigiert werden. Dieser Prozeß wird so lange wiederholt, bis alle Satzpositionszähler in aufsteigender Reihenfolge sind. Die Tabelle, in der die Wörter abgelegt sind, wird dabei *nicht* geändert. Geändert werden nur die Satzpositionszähler und die Wortzähler.

```
4000 FOR N=0 TO WC%-2
4010 IF WP(N,0)<WP(N+1,0) THEN NEXT N:GO
TO 4040
4020 D%=WP(N,0):WP(N,0)=WP(N+1,0):WP(N+1
,0)=D%
4030 D%=WP(N,1):WP(N,1)=WP(N+1,1):WP(N+1
,1)=D%:GOTO4000
```

Wenn die Zeichenketten nun entsprechend des revidierten Satzpositionszeigers ausgegeben werden, stehen sie in der gleichen Reihenfolge wie im Originalsatz und können dadurch sehr leicht wiedererkannt werden.

```
4040 PRINT:FOR N=0 TO WC%-1
4050 PRINT W$(WP(N,1));" ";
4060 NEXT N:PRINT
```

Alle Elemente der Satzpositionstabelle WP(N,0) und der Wortzähler WC% müssen vor einer neuen Eingabe auf 0 gesetzt werden.

```
4070 FOR N=0 TO 19
4080 WP(N,0)=0
4090 NEXT N
4100 WC%=0
4110 RETURN
```

KAPITEL 4

WIR GEBEN ANTWORTEN

SINNVOLLE ANTWORTEN

Bis jetzt haben wir uns nur darum gekümmert, Sätze zu decodieren, die in den Computer eingegeben wurden. Die Antworten, die er darauf gegeben hat, waren jedoch sehr begrenzt und simpel. Obwohl in einer Antwort oft viele der Originalwörter eines Satzes verwendet werden, wird bei einem wirklichen Dialog das Subjekt des Satzes in einer Antwort sinnentsprechend verändert.
Zum Beispiel wird auf die Eingabe:

ICH BRAUCHE RUHE

die bestätigende Antwortet erwartet:

DU BRAUCHE RUHE

und entsprechend:

DU BRAUCHST RUHE

sollte ergeben:

ICH BRAUCHST RUHE

Anmerkung: In der deutschen Sprache genügt das Austauschen des Subjektes nicht.
Wenn Sie diese Situation vom logischen Standpunkt überprüfen, werden Sie feststellen, daß es für jedes Eingabesubjekt ein entsprechendes Ausgabesubjekt gibt, und wir haben deshalb nur das Originalsubjekt abgetrennt und den Rest des Satzes an das entsprechende neue Subjekt angefügt.
"ICH" besteht aus drei Buchstaben, und wir können deshalb LEFT$(IN$,3) daraufhin abprüfen. Wenn es "ICH" war, können wir "DU" vor dem Rest der Eingabe RIGHT$(IN$,LEN(In$)−3) ausgeben.

```
10 INPUT IN$
30 IF LEFT$(IN$,3)="ICH" THEN PRINT "DU"
```

```
+RIGHT$(IN$,LEN(IN$)-3)
60 GOTO 10
```

In der gleichen Art können wir die ersten zwei Zeichen LEFT$(IN$,2) auf "DU" überprüfen und wenn notwendig durch "ICH" ersetzen.

```
50 IF LEFT$(IN$,2)="DU" THEN PRINT "ICH"
+RIGHT$(IN$,LEN(IN$)-2)
```

Wenn Sie dies mit einer Reihe von Sätzen ausprobieren, wird es bestens funktionieren, bis Sie z. B. folgenden Satz eingeben:

ICH BRAUCHE RUHE

worauf die ziemlich dumme Antwort kommt:

DU BRAUCHE RUHE

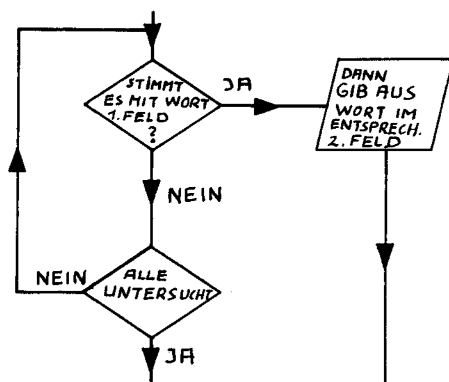
Aber wir können auch das in den Griff bekommen, indem wir nicht nur auf „ICH" und "DU" abprüfen, sondern auch auf "ICH BIN" und "DU BIST". Die Abprüfung darauf muß aber zuerst durchgeführt werden, und am Ende der Zeilen 20 und 40 muß ein Rücksprung auf die Zeile 10 durchgeführt werden, damit in den Zeilen 30 und 50 keine Übereinstimmung mit "ICH" oder "DU" gefunden wird.

```
20 IF LEFT$(IN$,7)="ICH BIN" THEN PRINT
"DU BIST"+RIGHT$(IN$,LEN(IN$)-7):GOTO10
40 IF LEFT$(IN$,7)="DU BIST" THEN PRINT
"ICH BIN"+RIGHT$(IN$,LEN(IN$)-7):GOTO10
```

Obwohl diese Methode funktioniert, wird es doch ein sehr umständliches Programm, da wir für jede Möglichkeit eine separate Programmzeile brauchen und die Länge des zu suchenden Wortes bzw. Ausdrucks jeweils berücksichtigen müssen. Wenn wir den Satz auf viele unterschiedliche Wörter abprüfen müssen, benutzen wir besser eine mehrdimensionale Tabelle, die in einer Schleife mit der Eingabe verglichen werden kann.
Am besten benutzen wir dafür eine zweidimensionale Tabelle IO$(n,m) bei der die erste Dimension jedes Elements, IO$(n,0), das Eingabewort oder den Eingabeausdruck darstellt, und die zweite Dimension, IO$(n,1), das entsprechende Ausgabewort bzw. den Ausgabeausdruck enthält. Um Fehler zu vermeiden, legen wir sie

paarweise in DATA-Anweisungen ab und lesen sie dann in die Tabelle ein. Mit dem folgenden Programm wird die Tabelle angelegt.

```
10 GOSUB 10000
10000 DIM IO$(5,1)
11000 DATA ICH,DU,DU,ICH,ICH BIN,DU BIST
,DU BIST,ICH BIN
12000 FOR N=0 TO 3
12010 READ IO$(N,0),IO$(N,1)
12030 NEXT N
13000 RETURN
```



Ablaufdiagramm 4.1 Gebrauch einer korrespondierenden Antwort

Wir führen wieder eine gleitende Untersuchung in einer Schleife durch, die im Moment nur das entsprechende Wort bzw. den entsprechenden Ausdruck ausgibt (siehe **Ablaufdiagramm 4.1**). Durch die gleitende Überprüfung sind wir in der Lage, auch in Ausdrücken eingebettete Leerzeichen zu berücksichtigen, da wir den Eingabebereich IN$ nicht vorher in "Wörter" aufgeteilt haben.

```
100 INPUT IN$
110 ST%=1
200 FOR M=0 TO 3
210 TA$=IO$(M,0):GOSUB 5000:SP%=IS%:IF S
P%>0 THEN PRINT IO$(M,1)
220 NEXT M
250 GOTO 100
```

Für den späteren Programmablauf ist es jedoch besser, die benötigte Antwort als eine neue Zeichenkette R1$ zu definieren und diese vor dem Verlassen der Prüfschleife auszugeben.

```
210 TA$=IO$(M,0):GOSUB 5000:SP%=IS%:IF S
P%>0 THEN R1$=IO$(M,1)
230 PRINT R1$
```

Um eine vollständige und sinnvolle Antwort zu bekommen, müssen wir nur noch den Rest des Originalsatzes R2$ unter Verwendung eines Leerzeichens hinten anfügen. Es ist nicht schwierig, diesen "Rest des Satzes" zu bekommen. Wir müssen nur die Endposition des Wortes (bzw. Ausdrucks) von der Länge des Satzes subtrahieren und diesen Wert in RIGHT$ benutzen. SP% gibt uns den Anfang des gefundenen Wortes (bzw. Ausdrucks) an, und da wir einen Bereich von der Länge dieses Wortes (bzw. Ausdrucks) in der ersten Dimension der Tabelle IO$(N,0) haben, brauchen wir nur SP%+LEN(IO$(N,0)) zu subtrahieren.

```
210 TA$=IO$(M,0):GOSUB 5000:SP%=IS%:IF S
P%=0 THEN 220
215 R1$=IO$(M,1):R2$=" "+RIGHT$(IN$,LEN(
IN$)-(SP%+LEN(IO$(M,0))))
230 PRINT R1$;R2$
```

ICH BIN SCHLAU

Der Computer wird mit Ihnen übereinstimmen:

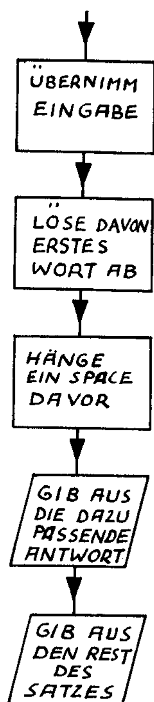
DU BIST SCHLAU

Wenn Sie jedoch RETURN noch einmal drücken, sagt er Ihnen immer noch, daß Sie schlau sind, was aber in diesem Fall bestimmt nicht mehr stimmt, da Sie die Bereiche IN$, R1$ und R2$ nicht "geleert" haben, bevor die neue Eingabe überprüft wird.

```
100 IN$="":INPUT IN$
240 R1$="":R2$=""
```

Bevor Sie sich aber zu schlau vorkommen, versuchen Sie folgendes:

WIR SIND DUMM



Ablaufdiagramm 4.2 Umfangreichere Antwort

und der Computer wird Sie mit folgender Antwort überraschen:

ICH M

Wenn Sie jedoch darüber nachdenken, wird Ihnen auffallen, daß sich eines unserer Schlüsselwörter in einem anderen Wort dieses Satzes verbirgt. Wenn es Ihnen nicht sofort klar ist, probieren Sie folgendes:

WIR SIND RICHTIG

und der Computer wird nur mit

DU IG

antworten.

Obwohl auf jedes Schlüsselwort abgeprüft wird, wird R1$ bei jeder Übereinstimmung neu gesetzt, so daß nur die letzte Übereinstimmung berichtet werden kann. Da jedoch der Satz auf jedes Schlüsselwort nur einmal abgeprüft wird, kann uns ein verstecktes "ICH" nur Probleme bringen, wenn es nicht das Schlüsselwort dieses Satzes ist.
Um auch mit diesem Problem fertig zu werden, müssen wir uns Gedanken über die Schlüsselwörter machen, die uns Probleme bringen können. Obwohl die Buchstabenkombination "ICH" recht oft vorkommt, ist sie jedoch sehr selten am Ende eines Wortes, und wir können deshalb abprüfen, ob ein Leerzeichen nach dem Schlüsselwort kommt. Da wir alle Schlüsselworte in der gleichen Art und Weise behandeln müssen, erweitern wir alle um ein Leerzeichen. Dies könnten wir durch eine Änderung der DATA-Anweisungen erreichen, aber wir sparen Speicherplatz, indem wir das Leerzeichen beim Erstellen der Tabellen hinzufügen. Es ist jedoch nicht nötig, auch bei den Antworten ein Leerzeichen anzufügen.

```
12020 IO$(N,0)=IO$(N,0)+" "
```

Beim Zusammenstellen der endgültigen Antwort R1$ müssen wir auch IN$ um ein Zeichen reduzieren, da das Leerzeichen nun ein Teil des Schlüsselwortes geworden ist.

```
215 R1$=IO$(M,1):R2$=" "+RIGHT$(IN$,LEN(
IN$)-(SP%+LEN(IO$(M,0)))+1)
```

Der Computer wird nun in der Lage sein, auf unsere Eingabe zu antworten.
Wenn jedoch das erste Schlüsselwort nicht am Anfang des Satzes steht, wird alles davor in der Antwort nicht mehr erscheinen.
Zum Beispiel wird die Frage:

HABE ICH ZEIT?

beantwortet mit:

DU ZEIT?

Der Computer gibt Ihnen auch sehr seltsame Antworten, wenn zwei echte Schlüsselworte vorhanden sind.
Die Frage:

WAS WENN DU UND ICH FALLEN?

wird beantwortet mit:

ICH UND DU FALLEN?

Aber es ist nicht so schwer, noch einige passende Schlüsselwörter in das Programm einzufügen, und man muß sich damit abfinden, daß einige Kombinationen einfach nicht akzeptiert werden. Um die Unterprogramme allgemein benutzen zu können, ist es besser, die Anzahl der Schlüsselwörter als Variable KW% zu definieren und diese dann anstatt der aktuellen Anzahl zu benutzen.

```
10 KW%=5:GOSUB 10000
200 FOR M=0 TO KW%
10000 DIM IO$(KW%,1)
11010 DATA WIR,WIR,SIE,SIE
12000 FOR N=0 TO KW%
```

ZEIGEN AUF ANTWORTEN

Bis jetzt hat der Computer nur wenig mehr Intelligenz als ein Papagei gezeigt, da er nur immer eine etwas modifizierte Version des Eingabetextes wiedergegeben hat. Wir müssen ihm also noch beibringen, einige logische Entscheidungen aufgrund der Eingabe zu machen, bevor er antwortet.

Die Anzahl der Subjekte SU%, der Verben VB% und der Antworten RP% wird als Variable definiert, so daß das Programm leicht erweitert werden kann, und die drei entsprechenden Tabellen werden aufbereitet. Da wir ein "Nullelement" in den Tabellen haben, sind alle Werte um eins kleiner als die Anzahl der Wörter. Die zweidimensionale Tabelle SU$(n,m) ist für die Subjekte in den Ein- und Ausgabesätzen zuständig. Die erste Dimension (n,0) enthält die in der Eingabe zu erkennenden Subjektworte bzw. Ausdrücke, und die zweite Dimension (n,1) enthält die entsprechenden Antworten. Die Tabelle VB$(n) enthält die zulässigen Verben, und die Tabelle RP$(n) enthält eine Serie von entsprechenden Antworten.

```
10 GOSUB 10000
10000 SU%=26:VB%=6:RP%=6
10010 DIM SU$(SU%,1)
10020 DIM VB$(VB%)
10030 DIM RP$(RP%)
```

SU$(N,0)	SU$(N,1)
ICH HABE	DU HAST
ICH MAG	DU MAGST
ICH BIN	DU BIST
ICH KENNE	DU KENNST
DU HAST	ICH HABE
DU MAGST	ICH MAG
DU BIST	ICH BIN
DU KENNST	ICH KENNE
ICH	DU
SIE HAT	SIE HAT
SIE IST	SIE IST
SIE MAG	SIE MAG
SIE	SIE
SIE WOLLEN	SIE WOLLEN
SIE SIND	SIE SIND
SIE MOEGEN	SIE MOEGEN
ER	ER
ER WILL	ER WILL
ER HAT	ER HAT
ER IST	ER IST
ER MAG	ER MAG
WIR HABEN	WIR HABEN
WIR MOEGEN	WIR MOEGEN
WIR SIND	WIR SIND
WIR WOLLEN	WIR WOLLEN
WIR	WIR
DU	ICH

Tabelle 4.1 Subjektgruppen in SU$(n,m)

Die ersten beiden DATA-Anweisungen enthalten paarweise gruppiert die Ein- und Ausgabesubjekte (siehe **Tabelle 4.1**), die in die Elemente der entsprechend dimensionierten Tabelle SU$(n,m) eingelesen werden. Da die Fürwörter (ICH, SIE, usw.) laufend mit anderen Wörtern zu Ausdrücken kombiniert werden (z. B. ICH HABE), werden diese Kombinationen ebenfalls in den DATA-Anweisungen angegeben. Sie werden so angegeben, daß die am meisten gebrauchten Kombinationen immer zuerst gefunden werden. Am Ende jeden Elementes wird noch ein Leerzeichen hinzugefügt, um teilweise Übereinstimmungen zu verhindern, und in der Ausgabe gleichzeitig ein Leerzeichen zu haben.

```
11000 DATA ICH HABE,DU HAST,ICH MAG,DU M
AGST,ICH BIN,DU BIST,ICH KENNE,DU KENNST

11010 DATA DU HAST,ICH HABE,DU MAGST,ICH
 MAG,DU BIST,ICH BIN,DU KENNST,ICH KENNE

11020 DATA ICH,DU,SIE HAT,SIE HAT,SIE IS
T,SIE IST,SIE MAG,SIE MAG,SIE,SIE
11030 DATA SIE WOLLEN,SIE WOLLEN,SIE SIN
D,SIE SIND,SIE MOEGEN,SIE MOEGEN,ER,ER
11040 DATA ER WILL,ER WILL,ER HAT,ER HAT
,ER IST,ER IST,ER MAG,ER MAG
11050 DATA WIR HABEN,WIR HABEN,WIR MOEGE
N,WIR MOEGEN,WIR SIND,WIR SIND
11055 DATA WIR WOLLEN,WIR WOLLEN,WIR,WIR
,DU,ICH
12000 FOR N=0 TO SU%
12010 READ SU$(N,0),SU$(N,1)
12020 SU$(N,0)=SU$(N,0)+" ":SU$(N,1)=SU$
(N,1)+" "
12030 NEXT N
```

Die nächste DATA-Anweisung enthält die Hauptverben, die danach in die Tabelle VB%$(n) eingelesen werden.

```
11060 DATA HASSE,LIEBE,TOETE,MAG NICHT,M
AG,FUEHLE,WEISS
12040 FORN=0 TO VB%
12050 READ VB$(N)
12060 NEXT N
```

Die letzte Gruppe von DATA-Anweisungen enthält die Antworten, die in die Tabelle RP$(n) eingelesen werden, bevor die Programmsteuerung über RETURN zum Hauptteil des Programmes zurückkehrt. Um das Programm auf dieser Stufe der Entwicklung möglichst einfach zu halten, wird in allen Antworten das Originalverb benutzt, obwohl man natürlich auch etwas anderes sagen könnte.

```
11070 DATA HASSE DICH WOHL EBENSO,LIEBE
DICH AUCH,TOETE DICH
11080 DATA MAG VIELE DINGE NICHT,MAG CHI
PS,FUEHLE MICH STARK,WEISS VIELE DINGE
12070 FORN=0 TO RP%
12080 READ RP$(N)
12090 NEXT N
13000 RETURN
```

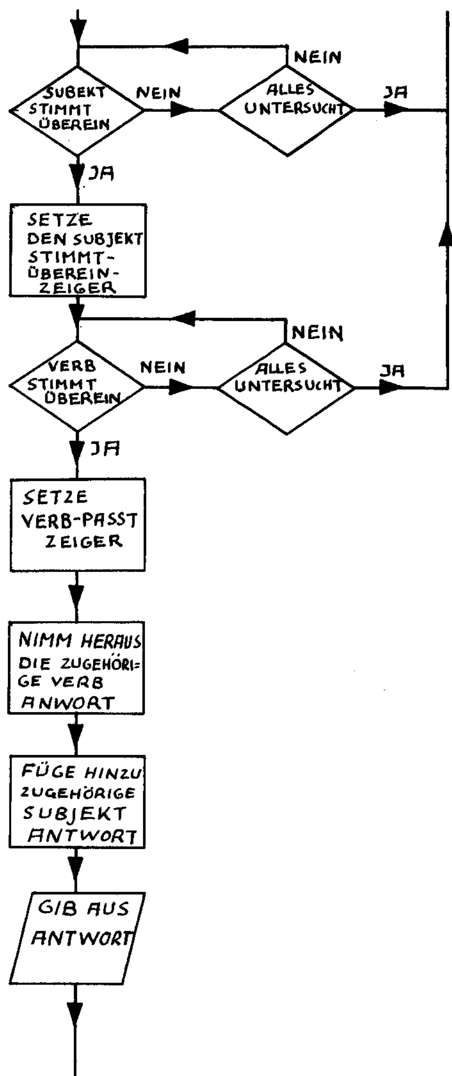
WIR SUCHEN ÜBEREINSTIMMUNGEN

Zuerst wird die Eingabe mit allen Subjekten in der ersten Dimension von SU$(n,n) verglichen (siehe **Ablaufdiagramm 4.3**). Wenn keine Übereinstimmung gefunden wird, wird eine neue Eingabe angefordert. Im Falle einer Übereinstimmung verweist die Variable SM% auf das Element, bei dem die Übereinstimmung stattgefunden hat. Die Variable IS% wird weiterhin dafür benutzt, die in dem Unterprogramm INSTR gefundene Position anzugeben.

```
200 FOR M=0 TO SU%
210 ST%=1:TA$=SU$(M,0):GOSUB 5000
220 IF IS%=0 THEN NEXT M:GOTO 100
230 SM%=M
```

Die Verbentabelle wird nun mit der Eingabe IN$ verglichen. Wenn kein Verb gefunden wird, wird die Eingabe zurückgewiesen, im anderen Fall wird die "Verb gefunden"-Variable VM% gesetzt.

```
240 FOR M=0 TO VB%
250 TA$=VB$(M):GOSUB 5000
260 IF IS%=0 THEN NEXT M:GOTO 100
270 VM%=M
```



Ablaufdiagramm 4.3 Setze Übereinstimmungszeiger

WIR GEBEN ANTWORT

Nachdem Objekt und Verb identifiziert wurden, können wir mit Hilfe von VM% die entsprechende Antwort aus der Antwortentabelle RP$(n) heraussuchen.

```
500 RL$=RP$(VM%)
```

Im einfachsten Fall brauchen wir dem entsprechenden Subjekt nur RL$ anzufügen, bevor wir es ausgeben.

```
520 RL$=SU$(SM%,0)+RL$
550 PRINT RL$
560 GOTO 100
```

Auf die Eingabe:

ICH HASSE COMPUTER

antwortet der Computer dann:

ICH HASSE DICH WOHL EBENSO

und:

ICH WEISS ALLES

ergibt:

ICH WEISS VIELE DINGE

WIR BENUTZEN ALTERNATIVE SUBJEKTE

Wenn Sie es jedoch bevorzugen, daß die Maschine mit Ihnen übereinstimmt und Sie nicht mehr ärgert, dann müssen Sie das zweite Element der Tabelle (das Gegenstück) zu RL$ hinzufügen.

```
520 RL$=SU$(SM%,1)+RL$
```

und:

ICH WEISS EINE MENGE

ergibt:

DU WEISS VIELE DINGE

Spaßeshalber können Sie auch das erste oder zweite Element 'Random' auswählen, so daß die Antwort nicht mehr voraussagbar ist.

```
510 RS%=INT(RND(1)+.5)
520 RL$=SU$(SM%,RS%)+RL$
```

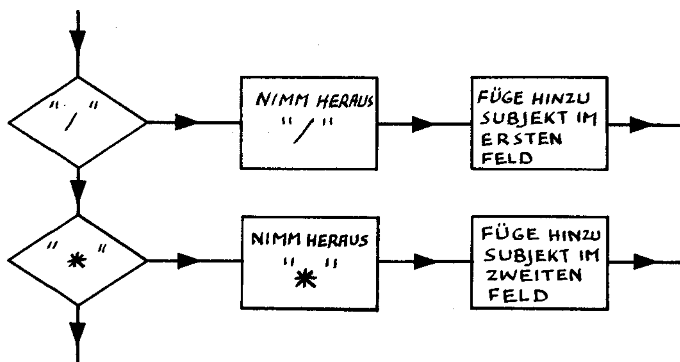
WIR SETZEN DAS SUBJEKT IN EINEN ZUSAMMENHANG

Es wäre aber insgesamt sehr viel besser, wenn wir das richtige Subjekt entsprechend des Inhaltes der Antwort auswählen könnten. Dafür ist es notwendig, in der Antwortentabelle Markierungen zu setzen. Für die Benutzung des Wortes in der ersten Dimension der Subjekttabelle benutzen wir einen Schrägstrich '/', und für die Benutzung des Wortes in der zweiten Dimension benutzen wir ein Sternchen '*'.

```
11070 DATA HASSE DICH WOHL EBENSO,LIEBE
DICH AUCH,TOETE DICH
11080 DATA MAG VIELE DINGE NICHT,MAG CHI
PS,FUEHLE MICH STARK,WEISS VIELE DINGE
```

Wir können jetzt das benötigte Element der Antwortentabelle RP$, das an der Stelle der Verbmarkierung VM% liegt, auf einen Schrägstrich untersuchen. Vor der Benutzung des INSTR-Unterprogramms weisen wir der Variablen IN$ den Wert dieses Elementes zu. Wenn ein Schrägstrich gefunden wird, wird der Inhalt der ersten Dimension der Subjekttabelle SU$(M%,0) der Antwort RL$ (jedoch ohne den Schrägstrich) hinzugefügt (siehe **Ablaufdiagramm 4.4**).

```
500 RL$=RP$(VM%)
510 IN$=RL$:ST%=1:TA$="/":GOSUB 5000
520 IF IS%>0 THEN 800
800 RL$=SU$(SM%,0)+RIGHT$(RL$,LEN(RL$)-1
)
810 GOTO 530
```



Ablaufdiagramm 4.4 Bringe Subjekt in Verbindung

Wenn jedoch kein Schrägstrich gefunden wurde, wird in einem zweiten Suchlauf der Stern gesucht. Ist die Suche erfolgreich, dann wird die zweite Dimension von SU$(SM%,1) in der gleichen Art angefügt.

```
530 ST%=1:TA$="*":GOSUB 5000
540 IF IS%>0 THEN 820
820 RL$=SU$(SM%,1)+RIGHT$(RL$,LEN(RL$)-1
)
830 GOTO 550
```

Nun wird:

ICH LIEBE MICH

ergeben:

ICH LIEBE DICH AUCH

aber:

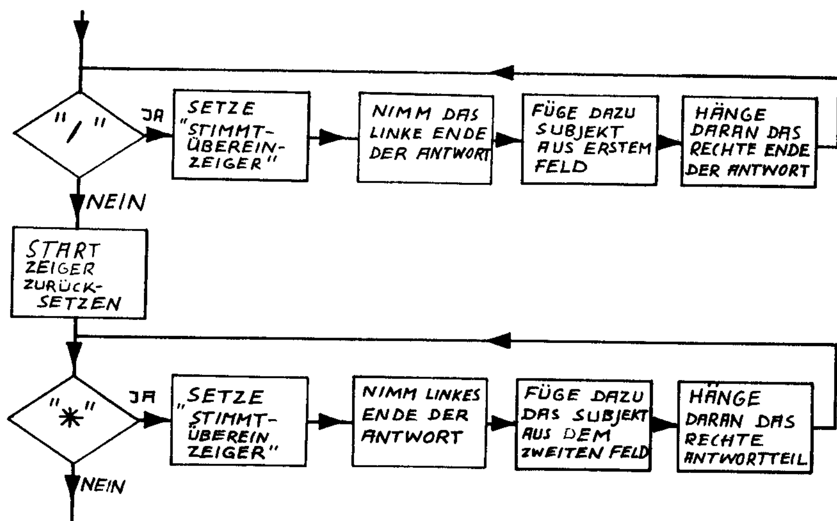
ICH FÜHLE MICH STARK

ergibt:

ICH FÜHLE ICH STARK

WIR FÜGEN IN SÄTZE EIN

Zur Vereinfachung fingen unsere Antworten immer mit dem Subjekt an, aber im wirklichen Leben ist das nicht immer so. Nachdem wir in unseren Antworten Markierungen haben, um anzuzeigen, welches Subjekt hinzugefügt werden muß, können wir diese ebenfalls dafür benutzen, um anzuzeigen, wo in der Antwort das Wort oder der Ausdruck eingesetzt werden muß. Zuerst müssen wir unsere DATA-Anweisung so erweitern, daß das einzufügende Wort (bzw. der Ausdruck) niemals am Anfang ist. Dadurch kann der Einfügungsprozeß offensichlich gemacht werden.



Ablaufdiagramm 4.5 In einen Satz einfügen

```
11070 DATA BEMERKST DU DASS /WOHL EBENSO
 DICH HASSE,GUT /LIEBE DICH AUCH
11080 DATA WENN /DICH NICHT ERST TOETE,S
OWAS *MAGST VIELE DINGE NICHT
11090 DATA NUN LIEBE /CHIPS,WARUM FUEHLS
T *DICH STARK,WIESO WEISST *ALLES
```

Im Prinzip wissen wir schon, wo das Wort eingefügt werden muß, da wir durch IS% wissen, wo der Schrägstrich oder Stern in der Antwort gefunden wurde. Wir müssen nur noch den Teil der Antwort vor der Markierung, LEFT$(RL$,IS%−1), nehmen

und dazu die korrekte Version von SU$(SM%,n) und den Rest der Antwort RIGHT$(RL$,LEN(RL$)−IS%) addieren.

```
800 RL$=LEFT$(RL$,IS%-1)+SU$(SM%,0)+RIGH
T$(RL$,LEN(RL$)-IS%)
810 GOTO 530
820 RL$=LEFT$(RL$,IS%-1)+SU$(SM%,1)+RIGH
T$(RL$,LEN(RL$)-IS%)
```

Jetzt:

ICH WILL IHN TÖTEN

ergibt:

WENN ICH DICH NICHT ERST TÖTE

und:

ICH MAG KEINE COMPUTER

ergibt:

NUN LIEBE ICH MAG CHIPS

Obwohl wir das Subjekt jetzt schon sehr viel besser in die Antwortsätze einfügen können, arbeiten wir immer nur mit einem Subjekt pro Satz. Eine weitere kleine Änderung wird es uns ermöglichen, jede Anzahl von Subjekten in einen Satz einzufügen. Wir müssen dazu nur so lange die Markierungen suchen, bis keine mehr zu finden sind. In Zeile 500 haben wir eine Startvariable ST% als 1 definiert, und danach wird die erste Markierung gesucht. Wenn sie gefunden ist, wird die Startvariable ST% auf einem Platz höher als die gefundene Markierung gesetzt. Nachdem RL% in Zeile 800 zusammengesetzt worden ist, springen wir zurück in Zeile 510, um nach weiteren Markierungen zu suchen. Wenn keine weiteren Übereinstimmungen mit der ersten Markierung mehr gefunden werden können, wird ST% wieder auf 1 zurückgesetzt. Nach der zweiten Markierung wird dann in der gleichen Art gesucht.

```
500 RL$=RP$(VM%):ST%=1
510 IN$=RL$:TA$="/":GOSUB 5000
520 IF IS%>0 THEN ST%=IS%+1:GOTO 800
525 ST%=1
```

```
530 IN$=RL$:TA$="*":GOSUB 5000
540 IF IS%>0 THEN ST%=IS%+1:GOTO 820
810 GOTO 510
830 GOTO 530

11070 DATA BEMERKST DU DASS /WOHL EBENSO
 DICH HASSE,GUT /LIEBE DICH AUCH
11080 DATA WENN /DICH NICHT ERST TOETE
11085 DATA JAJA /LIEBE CHIPS,/MAG VIELE
DINGE NICHT BESONDERS DICH
11090 DATA WARUM FUEHLST *DICH STARK,*DE
NKST *WEISST ALLES
```

Jetzt:

ICH WEISS ALLES

ergibt:

DU DENKST DU WEISST ALLES

und:

ICH HASSE COMPUTER

ergibt:

BEMERKST DU DASS ICH WOHL EBENSO DICH HASSE

PROBLEME MIT DEM SUBJEKT

Alles scheint wunderbar zu gehen, bis Sie folgende Eingabe machen:

ICH HASSE DICH

und folgende Antwort erscheint:

BEMERKST DU DASS ICH WOHL EBENSO DICH HASSE

Das Problem ist, daß wir die Suchroutine direkt nach der ersten Übereinstimmung verlassen, und obwohl wir das Subjekt "ICH" suchen, wir das Objekt "DICH" zuerst finden. Da "DICH" in der Subjekttabelle vor "Ich" kommt, wird es zuerst gefunden, obwohl es im Satz in Wirklichkeit später kommt.
Da wir nicht alle verwickelten Gedankengänge eines menschlichen Gehirns nachvollziehen können, müssen wir aus Gründen der Einfachheit von der Annahme ausgehen, daß das Subjekt immer vor dem Verb und das Objekt nach diesem kommt. Bis jetzt haben wir in dem Programm das Subjekt vor dem Verb gesucht; das müssen wir jetzt in umgekehrter Reihenfolge durchführen. Wenn ein Verb in der Eingabe gefunden wird, legen wir seine Position in IS% ab und weisen diesen Wert dann dem Verbpositionszeiger VP% zu.

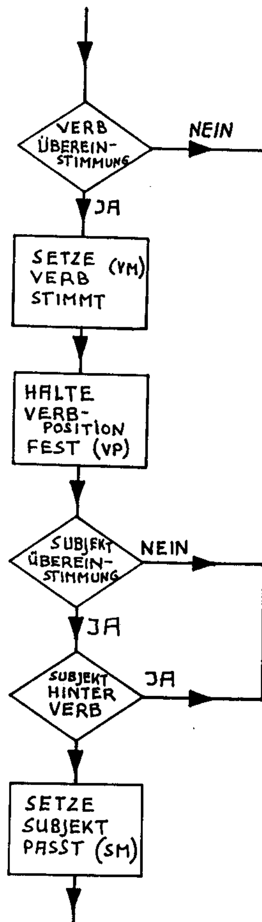
```
200 FOR M=0 TO VB%
210 ST%=1:TA$=VB$(M):GOSUB 5000
220 IF IS%=0 THEN NEXT M:GOTO 100
230 VM%=M:VP%=IS%
```

Wenn wir nun eine Übereinstimmung mit der Subjekttabelle gefunden haben, können wir die Position IS% mit dem Verbpositionszeiger VP% vergleichen und die Übereinstimmung zurückweisen, wenn das Subjekt hinter dem Verb positioniert ist (siehe **Ablaufdiagramm 4.6**).

```
240 FOR M=0 TO SU%
250 ST%=1:TA$=SU$(M,0):GOSUB 5000
260 IF IS%=0 THEN NEXT M:GOTO 100
270 IF IS%>VP% THEN NEXT M:GOTO 100
280 SM%=M
```

Wenn Sie keine Lust haben, diese Zeilen neu einzugeben, können Sie mit einer Anzahl von Sprüngen das gleiche erreichen.

```
140 GOTO 240
231 GOTO 500
271 GOTO 200
270 VM%=M:VP%=IS%
225 IF IS%>VP% THEN NEXT M:GOTO 100
```



Ablaufdiagramm 4.6 Abweisen von Objektpaarigkeiten

WIR WECHSELN DIE ZEIT

Um mit der Vergangenheit eines Verbs antworten zu können, fassen wir jeweils die Gegenwart und die Vergangenheit eines bestimmten Verbs in einer Zweiergruppe zusammen und legen diese dann in DATA-Anweisungen ab.

GEGENWART	VERGANGENHEIT
HASSE	HASSTE
LIEBE	LIEBTE
TOETE	TOETETE
MAG NICHT	MOCHTE NICHT
MAG	MOCHTE
FUEHLE	FUEHLTE
WEISS	WUSSTE

```
10000 SU%=26:VB%=13:RP%=6
11060 DATA HASSE,HASSTE,LIEBE,LIEBTE,TOE
TE,TOETETE,MAG NICHT,MOCHTE NICHT
11065 DATA MAG,MOCHTE,FUEHLE,FUEHLTE,WEI
SS,WUSSTE
```

Auf die Eingabe:

ICH HASSTE DICH

erfolgt die Antwort:

BEMERKST DU DASS ICH WOHL EBENSO DICH HASSE

Um jeweils mit der Vergangenheit des Verbs antworten zu können, dividieren wir die Verbvariable VM% durch zwei, womit wir die richtige Antwort erhalten.

```
230 VM%=M/2:VP%=IS%
```

KAPITEL 5

EXPERTENSYSTEME

Ein menschlicher Experte ist jemand, der eine Menge über einen bestimmten Bereich weiß, und der entsprechende Auskunft darüber geben kann. Die dafür notwendigen Kenntnisse können nur nach langer Tätigkeit in diesem Bereich erworben werden. Aus diesem Grunde sind wirkliche Experten unglücklicherweise sehr, sehr selten. Darüber hinaus sind sie meistens nicht zur Hand, wenn man sie zur Lösung eines Problems benötigt.

Aus diesem Grunde haben sich Wissenschaftler des Problems angenommen, Computer-Programme zu erstellen, die die Funktionen solcher menschlichen Experten übernehmen können. Solche Programme haben den Vorteil, daß durch deren Vervielfältigung unendlich viele Experten produziert werden können, die weder Kaffeepausen noch Schlaf noch Gehaltserhöhungen brauchen oder fordern. Natürlich kann der Computer nur den vorprogrammierten Instruktionen folgen, die ein Programmierer eingegeben hat. Interessanterweise haben Zukunftsroman-Autoren das Problem behandelt, wenn 'Superexperten' (wie HAL in "2001: Odyssee im Weltraum" oder Isaac Asimov's Positronic Roboter) mit Alternativen konfrontiert waren, die mit ihren primären Anweisungen in Konflikt gerieten, und dabei zwar keine Systemzusammenbrüche, jedoch Nervenzusammenbrüche "erlebten".

Bevor wir Programme für "Expertensysteme" schreiben können, müssen wir uns fragen, wie menschliche Experten arbeiten.

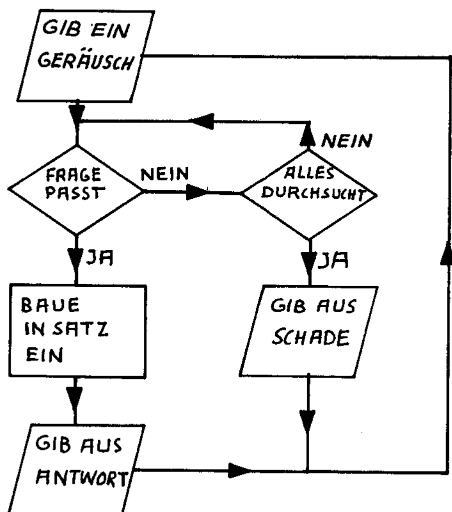
Lassen Sie uns zuerst die einfache Situation betrachten, in der es die Aufgabe eines Experten ist, die Antwort auf ein bekanntes Problem zu finden.

Zuerst sucht er sich Informationen über diese Aufgabe.

Danach vergleicht er diese mit den in seinem Gehirn gespeicherten Informationen und sucht nach Übereinstimmungen.

Letztlich berichtete er, ob er eine Übereinstimmung gefunden hat oder nicht.

Was wir brauchen, ist ein Datenverwaltungsprogramm, das versucht, die Eingabeinformation gegen die gespeicherte Information abzuprüfen (siehe **Ablaufdiagramm 5.1**). Ein benutzerfreundliches Programm würde natürlich mit der Umgangssprache arbeiten, aber aus Gründen der Einfachheit werden wir uns hier mit einer Eingabe in einem festen Format begnügen. Als Beginn werden wir versuchen, Tiere an den Geräuschen, die sie machen, zu erkennen. Wir erstellen zwei Tabellen: die Fragetabelle QU$(n) enthält die bekannten Geräusche, und jedes Element der Antwortentabelle AN$(n) enthält den Namen des entsprechenden Tieres.



Ablaufdiagramm 5.1 Ein einfacher 'Experte'

```
10 GOSUB 10000
10000 DIM QU$(4),AN$(4)
10010 DATA MIAU,KATZE,WAUWAU,HUND,MUH,KU
H,HUU,EULE,WIEHERN,PFERD
10020 FOR N=0 TO4:READ QU$(N),AN$(N):NEX
T N
10030 RETURN
```

Jetzt müssen wir nur noch nach einem Geräusch fragen und es mit dem Inhalt von QU$(n) vergleichen.

```
20 PRINT "WELCHE GERAEUSCHE MACHT DAS TI
ER ?"
30 INPUT IN$
40 FOR N=0 TO 4:IF IN$=QU$(N) THEN 100
50 NEXT N
60 PRINT "NA SOWAS, DIESES TIER KENNE IC
H NICHT."
70 GOTO 20
```

```
100 PRINT "EIN TIER, DAS ";QU$(N);" MACH
T IST EIN ";AN$(N)
110 GOTO 20
```

Vielleicht sollten wir schon hier sagen, daß unser Computer-Experte in diesem Bereich besser als sein menschlicher Kollege sein kann, da er keine subjektiven Urteile abgibt, keine Langeweile bekommt oder vergißt, alle Informationen in seinem Gehirn abzuprüfen. Auf der anderen Seite ist es nicht sehr gelehrt, wenn er nur antwortet "Ein Eule" usw. (Wir überlassen es Ihnen, eine Routine zu entwikkeln, die erkennt, ob es sich um ein männliches, weibliches oder sächliches Objekt handelt.)

WIR LERNEN ZU ENTSCHEIDEN

Das letzte Beispiel war sehr einfach. Es wurde nur eine Frage gestellt, und es gab nur eine mögliche Antwort. In Wirklichkeit müssen wir in der Lage sein, sehr viel schwierigere Probleme zu lösen, bei denen die Antwort nicht durch eine Serie von Fragen gefunden werden kann. Zum Beispiel, was sollte ein Experte tun, wenn er sein Auto starten will, und nichts passiert?
Es könnte dafür eine Anzahl von Gründen geben:

LEERE BATTERIE
BATTERIEKABEL NICHT ANGESCHLOSSEN
FEHLERHAFTER ZUENDSCHALTER
FESTGEFRESSENER ANLASSER
ZERSTOERTER ANLASSER
ZERSTOERTE ZUENDSPULE

Um den Fehler zu finden, sollte er mehrere Überprüfungen in einer logischen Folge durchführen. Zuerst sollte er überprüfen, ob das Problem ein nicht arbeitender Anlasser ist:

BRENNT ZUENDKONTROLLICHT? (J/N)

Wenn die Antwort darauf "N" ist, dann liegt keine Spannung am Zündschalter, so daß nur eine der drei obigen Fehlermöglichkeiten besteht. Wir können die Anzahl der Fehlermöglichkeiten weiter verringern, indem wir herausfinden, ob die Beleuchtung arbeitet:

BRENNEN LAMPEN RICHTIG? (J/N)

Wenn die Antwort "J" ist, kann die Batterie nicht leer sein und muß einwandfrei mit dem Lichtschalter verbunden sein. Jetzt kann man davon ausgehen, daß der Zündschalter defekt ist, und es ist möglicherweise zu empfehlen, ihn zu ersetzen.

BAUE DEN ZUENDSCHALTER AUS

Wenn die Lampen nicht brennen, sollten die Verbindungen überprüft werden.

SIND BATTERIEKABEL ANGESCHLOSSEN? (J/N)

Wenn die Antwort "J" ist, dann ist die Batterie leer, und Sie müssen sie laden (oder das Auto schieben).

LADE BATTERIE ODER LASS DICH SCHIEBEN

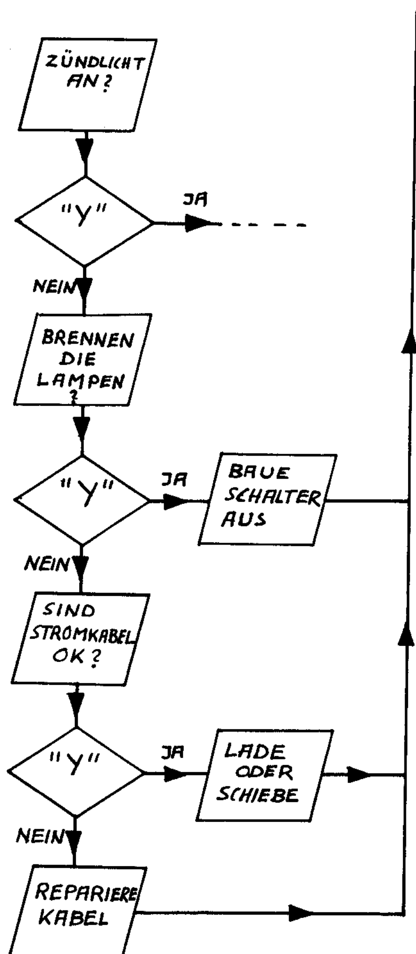
In derselben Art und Weise kann man mit einer Folge von Überprüfungen das Problem lösen, wenn Spannung da ist, aber der Anlasser nicht funktioniert (die letzten drei Möglichkeiten).

Diese Entscheidungsstruktur läßt sich am einfachsten durch eine Folge von "IF THEN"-Tests programmieren (siehe **Ablaufdiagramm 5.2**).

```
10 PRINT "FEHLERDIAGNOSE"
20 PRINT
30 PRINT "BRENNT ZUENDKONTROLLICHT  (J/N
) ?"
40 INPUT IN$
50 IF IN$="J" THEN 180
60 PRINT "BRENNEN LAMPEN RICHTIG (J/N) ?
"
70 INPUT IN$
80 IF IN$="N" THEN 110
90 PRINT "BAUE DEN ZUENDSCHALTER AUS"
100 INPUT IN$
110 PRINT "SIND BATTERIEKABEL ANGESCHLOS
SEN (J/N) ?"
120 INPUT"<RETURN>TASTE:";T$:RUN
130 IF IN$="J" THEN 160
140 PRINT "MACHE BATTERIEKABEL FEST"
```

```
150 INPUT"<RETURN>TASTE:";T$:RUN
160 PRINT "LADE BATTERIE ODER LASS DICH
ANSCHIEBEN"
170 INPUT"<RETURN>TASTE:";T$:RUN
180 REM --- ETC ---
```

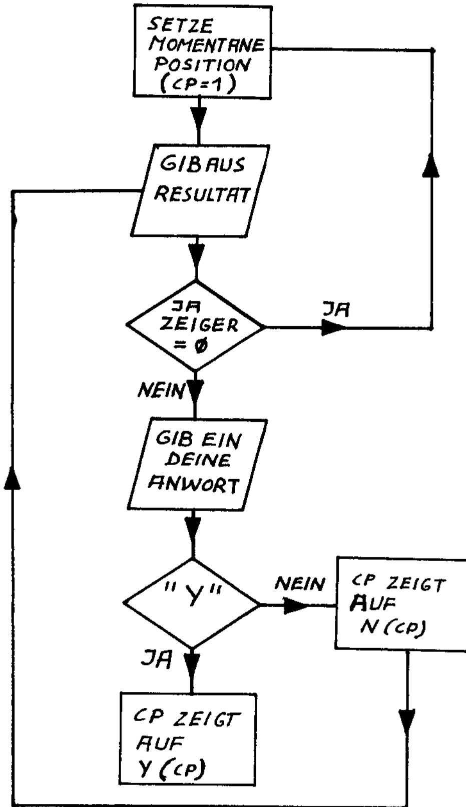
Ein solches Programm ist relativ leicht zu schreiben, aber es ist sehr ineffizient, je länger und komplizierter es wird.



Ablaufdiagramm 5.2 Ein sich verzweigender 'Experte'

WIR SUCHEN EINEN BESSEREN WEG

Ein sehr viel effizienterer Weg zur Lösung dieses Problems ist es, den Text in Tabellen abzulegen, um mit Zeigern zu arbeiten, die Sie zur nächsten Frage oder Antwort führen, abhängig davon, ob die Antwort auf die laufende Frage "Ja" oder "Nein" war (siehe **Ablaufdiagramm 5.3**).



Ablaufdiagramm 5.3 Auf die nächste Antwort zeigend

Das allgemeine Format zur Beschreibung der Entscheidungspunkte in den DATA-Anweisungen ist folgendermaßen:

(TEXT), (Zeiger für 'JA'), (Zeiger für 'NEIN')

Die erste Frage war:

BRENNT ZUENDKONTROLLICHT? (J/N) . . . 1

Wenn die Antwort "N" war, müssen Sie die zweite Frage stellen:

BRENNT ZUENDKONTROLLE HELL? (J/N) . . . 2

Andererseits müssen Sie mit den anderen Teilen Ihrer Fehlerdiagnose fortfahren, die wir hier nicht weiter verfolgt haben. Es wäre Frage 7.
Wir müssen insgesamt drei Tabellen erstellen. OP$(n) enthält den Ausgabetext, Y(n) die Zeiger für "Ja" und N(n) die Zeiger für "Nein". Um eine leichte Änderung des Programms zu ermöglichen, benutzen wir die Variable NP für die Anzahl dieser Punkte. Die DATA-Anweisungen werden in Dreiergruppen gelesen und jeweils eines in jedes Element dieser Tabellen abgelegt. Um das mögliche Ende einer Diagnose anzuzeigen, werden die Ja- und Nein-Zeiger auf Null gesetzt.

```
10 GOSUB 10000
10000 NP=7
10010 DIM OP$(NP),Y(NP),N(NP)
11000 DATA "BRENNT ZUENDKONTROLLICHT",7,
2
11010 DATA "BRENNEN LAMPEN RICHTIG",3,4
11020 DATA "ERSETZE ZUENDSCHALTER",0,0
11030 DATA "BATTERIE RICHTIG ANGESCHLOSS
EN",5,6
11040 DATA "BATTERIE LADEN ODER SCHIEBEN
",0,0
11050 DATA "UEBERPRUEFE BATTERIEVERBINDU
NG",0,0
11060 DATA "PROGRAMMERWEITERUNGEN",0,0
12000 FOR N=1 TO NP
12010 READ OP$(N),Y(N),N(N)
12020 NEXT N
13000 RETURN
```

Der Programmablauf ist sehr einfach. Mit Hilfe eines Zeigers CP wird auf die laufende Position der Tabelle gezeigt; am Anfang enthält er eine 1, und der erste Text wird ausgegeben. Wenn wir schon eine Endposition Y(CP)=0 (was ziemlich unwahrscheinlich ist) erreicht haben, dann wird CP auf 1 zurückgesetzt, so daß die Abfragen wieder neu beginnen können. Wenn jedoch ein realistischer Zeiger

vorhanden ist, dann wird die entsprechende Antwort abgefragt. Wenn die Antwort "J" ist, wird CP auf den Wert in dem entsprechenden Element der Y(n)-Tabelle gesetzt; im anderen Fall bekommt es den Wert der N(n)-Tabelle.

```
20 CP=1
30 PRINT OP$(CP)
40 IF Y(CP)=0 THEN 20
50 INPUT IN$
60 IF IN$="J" THEN CP=Y(CP):GOTO 30
70 CP=N(CP)
80 GOTO 30
```

WIR ARBEITEN PARALLEL

Eine Alternative zu der oben beschriebenen sequentiellen Entscheidungsmethode ist die parallele Methode, bei der vor einer Entscheidung erst alle möglichen Fragen gestellt werden. Diese Methode dauert im allgemeinen länger, als wenn man einer effizienten Baumstruktur folgt, aber die Wahrscheinlichkeit einer korrekten Antwort ist hier größer, da keine notwendigen Vergleiche ausgelassen werden.
Lassen Sie uns überlegen, wie wir zwischen verschiedenen Fortbewegungsformen unterscheiden können.
Wir wollen acht Eigenschaften berücksichtigen, und mit 1 oder 0 ihr Vorhandensein oder Nichtvorhandensein bei jeder unserer fünf Transportmöglichkeiten angeben (siehe **Tabelle 5.1**). Bei genauer Betrachtung werden Sie feststellen, daß das Ergebnis für alle unterschiedlich ist, so daß es möglich sein muß, zwischen diesen zu unterscheiden.

	Fahrrad	Auto	Eisenbahn	Flugzeug	Pferd
Räder	1	1	1	1	0
Flügel	0	0	0	1	0
Motor	0	1	1	1	0
Reifen	1	1	0	1	0
Schienen	0	0	1	0	0
Fenster	0	1	1	1	0
Ketten	1	0	0	0	0
Lenkrad	1	1	0	1	1

Tabelle 5.1 An- oder Abwesenheit der Eigenschaften

Wir geben diese Werte als DATA-Anweisungen ein und speichern sie in eine zweidimensionale Tabelle FE(n,n) (die damit dieses Ergebnismuster enthält) und in eine Tabelle mit den Namen der Objekte OB$(n).

```
10 GOSUB 10000
10000 NP=7
10010 DIM OB$(5),FE(5,8)
11000 DATA FAHRRAD,1,0,0,1,0,0,1,1
11010 DATA AUTO,1,0,1,1,0,1,0,1
11020 DATA EISENBAHN,1,0,1,0,1,1,0,0
11030 DATA FLUGZEUG,1,1,1,1,0,1,0,1
11040 DATA PFERD,0,0,0,0,0,0,0,1
12000 FOR N=1 TO 5
12010 READ OB$(N)
12020 FOR M=1 TO 8
12030 READ FE(N,M)
12040 NEXT M,N
13000 RETURN
```

Wir können nun fragen, ob die erste Möglichkeit gegeben ist oder nicht und können damit ausdrücken, welche Transportart hier möglich ist (siehe **Ablaufdiagramm 5.4**).

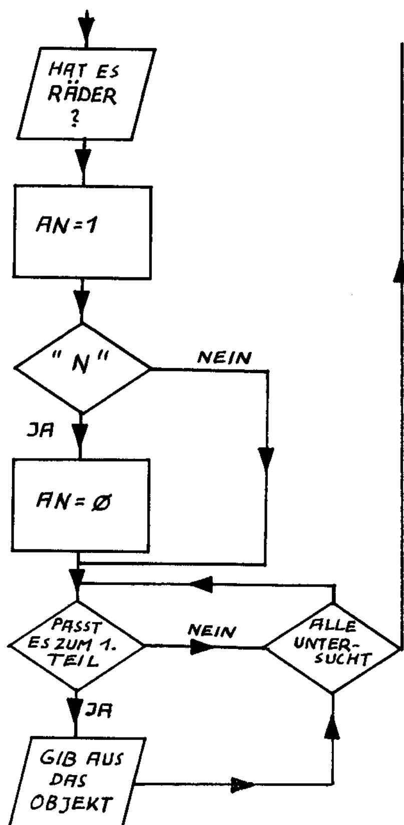
```
100 PRINT"█ HAT ES RAEDER"
500 INPUT IN$
510 AN=1:IF IN$="N" THEN AN=0
520 FOR N=1 TO 5
530 IF FE(N,1)=AN THEN PRINT OB$(N)
540 NEXT N:END
```

In diesem Fall würde die Antwort "J" folgende Ausgabe ergeben:

FAHRRAD
AUTO
EISENBAHN
FLUGZEUG

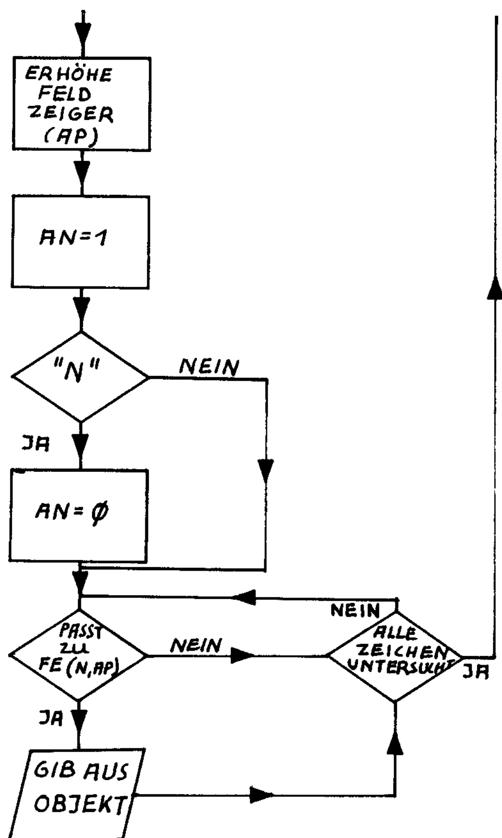
und die Antwort "N" würde nur folgende Ausgabe ergeben:

PFERD



Ablaufdiagramm 5.4 Parallele Annäherung

Dies zeigt ziemlich klar die möglichen Nachteile der Parallelmethode, obwohl wir gerade gezeigt haben, daß nur ein Pferd keine Räder hat. Das Programm erwartet jedoch, daß wir noch weitere Fragen stellen, bevor es sich selbst entscheidet. Dies ist nicht so dumm, wie es auf den ersten Blick erscheint, da bei der Antwort "J" auf die nächste Frage ("Hat es Flügel?") der Computer logischerweise ablehnt, zu glauben, daß es fliegende Pferde gibt.
Wenn wir den eigentlichen Vergleichsteil als Unterprogramm schreiben, können wir es für die Untersuchung aller acht Möglichkeiten benutzen. Wir müssen einige kleine Modifikationen durchführen und einen Tabellenzeiger AP einführen, der bei jedem Prüfzyklus hochgezählt wird, um das nächste Element der Eigenschaftstabelle FE(N,AP) abprüfen zu können (siehe **Ablaufdiagramm 5.5**).



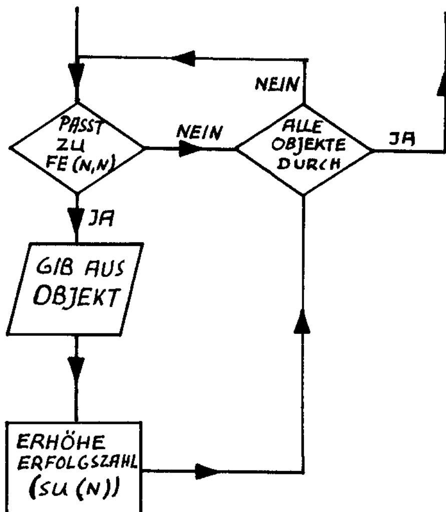
Ablaufdiagramm 5.5 Merkmale in Folge untersuchen

```
100 PRINT"␣ HAT ES RAEDER"
110 GOSUB 500
120 PRINT"HAT ES FLUEGEL"
130 GOSUB 500
140 PRINT"HAT ES EINEN MOTOR"
150 GOSUB 500
160 PRINT"HAT ES REIFEN"
170 GOSUB 500
180 PRINT"BENOETIGT ES SCHIENEN"
190 GOSUB 500
```

```
200 PRINT"HAT ES FENSTER"
210 GOSUB 500
220 PRINT"HAT ES EINE KETTE"
230 GOSUB 500
240 PRINT"IST ES LENKBAR"
250 GOSUB 500
400 PRINT
410 RUN
510 AP=AP+1:AN=1:IF IN$="N" THEN AN=0
530 IF FE(N,AP)=AN THEN PRINT OB$(N)
560 RETURN
```

WIR ZÄHLEN DIE RICHTIGEN ANTWORTEN

Das vorherige Programm gibt jeweils für jede Frage eine Liste von Antworten aus, aber es kann uns nicht sagen, welche Gruppe von Informationen die gültige Gesamtantwort auf alle Fragen ist. Wir können uns jedoch einen "Trefferzähler" erstellen, der uns angibt, wie gut unsere Antworten waren. Dazu erstellen wir uns eine Erfolgstabelle SU(n) für jedes Objekt, die jeweils bei einer richtigen Antwort FE(N,AP)=AN hochgezählt wird (siehe **Ablaufdiagramm 5.6**).



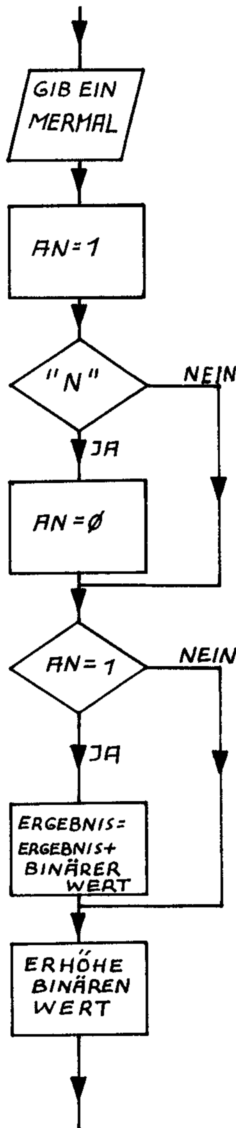
Ablaufdiagramm 5.6 Erfolg zählen

```
260 PRINT
270 PRINT"PUNKTZAHL"
280 PRINT
300 FOR N=1 TO 5
310 PRINT OB$(N),SU(N)
320 NEXT N
530 IF FE(N,AP)=AN THEN PRINT OB$(N):SU(
N)=SU(N)+1
10010 DIM SU(5)
```

Wenn wir eine vollkommene Übereinstimmung erreicht haben, wird der Erfolgszähler SU(n) gleich 8 sein. Wenn einer oder mehrere Punkte nicht korrekt waren, wird der Zähler entsprechend verringert. Diese Art des Zählens ist besonders nützlich, wenn die korrekte Antwort auf die Fragen mehr eine Sache der Einstellung als der wirklichen Tatsachen ist (z. B. ist ein Pferd wirklich steuerbar?). Die höchste erreichte Punktzahl zeigt wahrscheinlich sowieso auf die richtige Antwort, vorausgesetzt, daß eine gleichmäßige Gewichtung auf jede richtige Antwort erfolg.

WIR BENUTZTEN JETZT BITS

Es wird Ihnen bestimmt aufgefallen sein, daß wir gerade acht Eigenschaften zum Vergleich benutzt haben. Dies war beabsichtigt, weil acht Bits ein Byte ergeben. Wenn wir davon ausgehen, daß jede Möglichkeit durch ein Bit repräsentiert wird (siehe **Tabelle 5.2**), anstatt dafür einen absoluten Wert zu benutzen, dann kann jedes Objekt anstelle von acht separaten Werten durch eine einzelne Dezimalzahl, bestehend aus der Summe der Bits, beschrieben werden. Bei der Umrechnung in Dezimalzahlen beginnen wir mit dem niedrigstwertigen Bit in der obersten Reihe bei "Räder". Die einzelnen Möglichkeiten werden dann von oben nach unten mit den Werten von 1, 2, 4, 8 bis 128 dezimal dargestellt.



Ablaufdiagramm 5.7 Binäres Ergebnis erzeugen

	Fahrrad	Auto	Eisenbahn	Flugzeug	Pferd
Räder	1	1	1	1	0
Flügel	0	0	0	2	0
Motor	0	4	4	4	0
Reifen	8	8	0	8	0
Schienen	0	0	16	0	0
Fenster	0	32	32	32	0
Ketten	64	0	0	0	0
Lenkrad	128	128	0	128	128
Summe	201	173	53	175	128

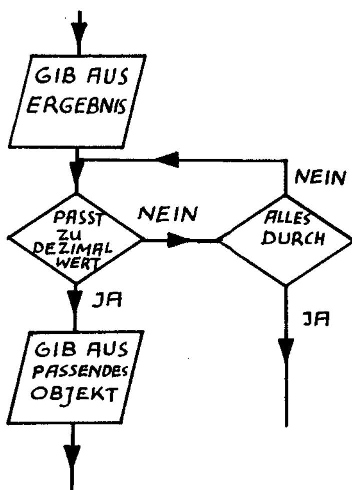
Tabelle 5.2 Binär gewichtete Eigenschaften

Es ist nicht sehr schwierig, unsere Punktzahl zu berechnen. Der Dezimalwert unserer Binärzahl BV verdoppelt sich bei jedem Schritt nach unten, und wir müssen den Wert bei jedem "Ja"-Fall zur Gesamtpunktzahl addieren (AN=1, siehe **Ablaufdiagramm 5.7**).

Wir brauchen nur die Punktzahl SU durch Addieren der Werte in jedem "Ja"-Fall zu aktualisieren. Es ist dazu nicht notwendig, jedesmal jedes Tabellenelement zu überprüfen oder sogar eine zweidimensionale Tabelle anzulegen. Die einzigen DATA, die wir benötigen, sind die Gesamtdezimalwerte jedes Objektes DV(n). Nachdem alle Fragen gestellt worden sind, überprüfen wir diese gegen den Dezimalwert aus den "Ja/Nein"-Anworten SU (siehe **Ablaufdiagramm 5.8**). Das beste ist, Sie löschen alles nach der Programmzeile 260 und beginnen vollkommen neu.

```
260 PRINT
270 PRINT "PUNKTZAHL";SU:GOSUB20000
280 PRINT
300 FOR N=1 TO 5
310 IF DV(N)=SU THEN PRINT,OB$(N):GOTO 4
00
320 NEXT N
330 PRINT,"KEIN OBJEKT GEFUNDEN"
400 PRINT
410 RUN
500 INPUT IN$
510 AN=1:IF IN$="N" THEN AN=0
520 IF AN=1 THEN SU=SU+BV
```

```
530 BV=BV+BV
540 RETURN
10000 DIM OB$(5),DV(5)
10010 BV=1
11000 DATA FAHRRAD,201
11010 DATA AUTO,173
11020 DATA EISENBAHN,53
11030 DATA FLUGZEUG,175
11040 DATA PFERD,128
12000 FOR N=1 TO 5
12010 READ OB$(N),DV(N)
12020 NEXT N
13000 RETURN
```



Ablaufdiagramm 5.8 Auf dezimale Werte untersuchen

Diese Vorgehensweise erspart uns bestimmt eine Menge Speicherplatz und Zeit, da jedes Tabellenelement mehrere Bytes benötigt und vor dem Vergleichen erst gesucht werden muß. Sie ist also besonders sinnvoll, wenn man große Informationsmengen bearbeiten muß. Andererseits müssen Sie jedoch alle Dezimalwerte des Bit-Musters berechnen, bevor Sie sie benutzen können, und es gibt Ihnen keinen Hinweis, wenn keine komplette Übereinstimmung gefunden ist. Sie können nicht einfach den Dezimalwert übernehmen, der am nächsten ist, da der Wert jeder

richtigen Antwort nur von seiner Position in der Tabelle abhängt. Natürlich können Sie die Berechnungen selbst machen, aber Sie können es auch dem Computer überlassen. Wenn Sie das Bit-Muster als eine Zeichenkette I$ eingeben, ist es relativ einfach, den Dezimalwert DV auszurechnen. Sie müssen nur jedes einzelne Zeichenscheibchen MID$(I$,N,1) mit "1" vergleichen und bei Übereinstimmung den Wert der Variablen BD addieren.

```
20000 BD=1:INPUT I$
20010 FOR N=1 TO 8
20020 IF MID$(I$,N,1)="1" THEN DV=DV+BD
20030 BD=BD+BD
20040 NEXT N
20050 PRINT DV
20060 RUN:
```

KAPITEL 6

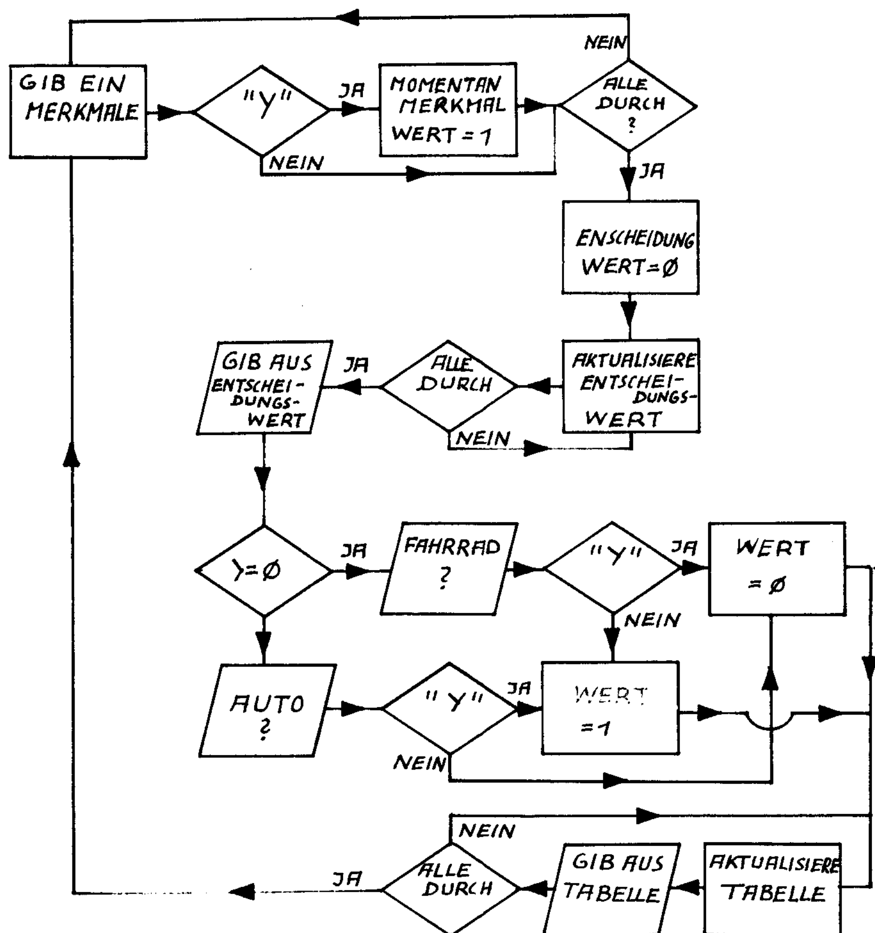
DAS "EXPERTENSYSTEM" LERNT VON SICH SELBST

Obwohl die bis jetzt beschriebenen Expertensysteme einwandfrei arbeiten, sind sie immer noch davon abhängig, daß Sie Ihnen als Basis Ihrer Entscheidungen die korrekten Regeln im voraus mitteilen. Das macht das Ganze natürlich sehr umständlich.
Es ist jedoch möglich, ein Expertenprogramm zu erstellen, das von seinen eigenen Fehlern lernen kann und seine eigenen Entscheidungsregeln erstellt. Voraussetzung ist, daß Sie ihm mitteilen können, wann (aber nicht unbedingt wo) es sich falsch verhält. Dies ist natürlich ein ganz erheblicher Vorteil, insbesondere in den Fällen, in denen Sie sich selbst nicht sicher über die korrekten Regeln sind. Dazu beginnen wir mit einer Gruppe von Eigenschaften, die uns erlauben, zwischen verschiedenen Objekten zu unterscheiden, jedoch ohne bereits die "Ja/Nein-Regeln" für diese Eigenschaften festzulegen. Wir benutzen das Programm, um diese Entscheidungsregeln festzulegen.
Wir arbeiten wieder mit dem uns bereits bekannten Transportproblem und definieren uns am Anfang einige Variablen. FE% ist die Anzahl der zu berücksichtigenden Eigenschaften (8), FE$(n) ist eine Tabelle mit den Namen dieser Eigenschaften, FV(n) ist eine Tabelle, die die Entscheidungswerte (0 oder 1) für jede Eigenschaft als Eingabe für jeden Entscheidungsfall enthalten wird, und RU(n) ist eine Tabelle, die die Summe sämtlicher Entscheidungswerte einer Entscheidungsregel für jede Eigenschaft enthalten wird.

```
10 GOSUB 10000
10000 FE%=8
10010 DIM FE$(FE%),FV(FE%),RU(FE%)
10020 FOR N=1 TO FE%
10030 READ FE$(N)
10040 NEXT N
11000 DATA RAEDER,FLUEGEL,MOTOR,REIFEN,S
CHIENEN,FENSTER,KETTEN,STEUERBAR
12000 RETURN
```

Jede Eigenschaft wird hierbei in Betracht gezogen (siehe **Ablaufdiagramm 6.1**). Zuerst wird der laufende Eigenschaftswert FV(n) für diesen Zyklus auf Null gesetzt, und danach wird für jeden Entscheidungspunkt vom Benutzer eine "Ja/Nein"-

Eingabe IN$ abgefragt. Wenn IN$ "Ja" ist, wird das Eigenschaftswertelement FV(n) auf 1 gesetzt; anderenfalls verbleibt es 0. Dadurch ergibt sich ein Muster, das das Objekt in der Tabelle FV(n) durch "0" und "1" beschreibt.



Ablaufdiagramm 6.1 Lernen zwischen 2 Objekten zu unterscheiden

```
60 FOR N=1 TO FE%
70 FV(N)=0
80 PRINT FE$(N);"  ";
```

```
90 GET IN$:IF IN$="" THEN 90
100 PRINT IN$
110 IF IN$="J" THEN FV(N)=1
120 NEXT N
```

Bevor Sie starten, wird die Entscheidungsvariable DE% auf 0 gesetzt. Sie wird in einer Schleife jedesmal neu aus ihrem laufenden Wert plus jedem der eingegebenen Eigenschaftswerte FV(n), multipliziert mit dem jeweiligen Wert der Entscheidungsregel RU(n) berechnet.

```
125 DE%=0
130 FOR N=1 TO FE%
150 DE%=DE%+FV(N)*RU(N)
160 NEXT N
170 PRINT "DE%=";DE%
```

WAS IST WAS?

Am Anfang betrachten wir eine sehr einfache Situation, bei der wir nur zwischen zwei Möglichkeiten auswählen, einem Fahrrad oder einem Auto. Wir beginnen damit, daß wir eine ziemlich willkürliche Entscheidung zwischen diesen beiden treffen. Wenn der Endwert DE% gleich oder größer als Null ist, dann ist es ein Auto. Es macht nichts aus, ob diese Annahme wirklich wahr ist, da sich unser System selbst korrigieren wird. Wenn das Programm, basierend auf dem Wert von DE%, eine Entscheidung getroffen hat, fragt es den Benutzer um Zustimmung oder Ablehnung dieser Entscheidung.

```
180 IF DE%>=0 THEN PRINT"IST ES EIN FAHR
RAD ";:INPUT IN$:GOTO200
190 IF DE%<0 THEN PRINT"IST ES EIN AUTO
";:INPUT IN$:GOTO 220
```

Abhängig von der Entscheidung des Computers sind drei verschiedene Reaktionen möglich. Bei korrekter Entscheidung ist keine Reaktion notwendig (die Gewichtungsvariable WT% wird 0 gesetzt), und das Programm springt für einen weiteren Versuch zurück. Wenn DE% >=0 war, aber der Computer unrecht hatte, dann wird die Gewichtungsvariable WT% auf minus eins gesetzt, während im letzten Fall, wenn DE% < 0 war und der Computer unrecht hatte, wird WT% auf plus eins gesetzt.

```
200 IF IN$="J" THEN WT%=0:GOTO240
210 WT%=-1:GOTO 240
220 IF IN$="J" THEN WT%=0:GOTO240
230 WT%=1
```

Mit der Gewichtungsvariablen werden dann die Werte der Regeltabelle RU(n) modifiziert. Wenn sie zu hoch waren, werden sie erniedrigt, und wenn sie zu niedrig waren, werden sie erhöht.

```
240 FOR N=1 TO FE%
250 RU(N)=RU(N)+FV(N)*WT%
260 PRINT RU(N),
270 NEXT N
280 PRINT
290 GOTO 60
```

An dem folgenden Beispiel kann man diese Vorgehensweise sehr gut erkennen. Starten Sie das Programm durch RUN und geben dann die folgenden Eingaben ein. (Um das gewünschte Bildschirmformat zu erreichen, geben Sie bitte alle Eingaben in der angegebenen Weise korrekt ein.)
Zuerst geben Sie bitte folgende Werte ein:

```
RÄDER J        FLÜGEL N     MOTOR N     REIFEN J
SCHIENEN N     FENSTER N    KETTEN J    STEUERBAR J
```

Das Programm wird mit einem Entscheidungswert DE% = Null antworten, da dies der Anfangswert ist, und bis jetzt keine Modifizierungen durchgeführt wurden:

```
DE%=0
```

Da DE% 0 ist, geht das System davon aus, daß es sich um ein Fahrrad handelt und bittet um Bestätigung, die natürlich mit "Ja" gegeben wird.

```
IST ES EIN FAHRRAD ? J
```

Der Ausdruck der Regeltabelle RU(n) bestätigt uns, daß sich keine Änderungen von 0 bis 1 ergeben haben, da die richtige Antwort direkt durch reinen Zufall erreicht wurde:

0	0	0	0
0	0	0	0

Nun versuchen Sie bitte folgende Eingabe, die ein Auto beschreibt:

RÄDER J	FLÜGEL N	MOTOR J	REIFEN J
SCHIENEN N	FENSTER J	KETTEN N	STEUERBAR J

DE% ist immer noch Null, so daß die falsche Folgerung gezogen und damit die falsche Frage gefragt wird, auf die die Antwort natürlich "N" sein muß:

DE%=0
IST ES EIN FAHRRAD ? N

Da ein Fehler gemacht wurde, wird die Entscheidungsregel durch Subtraktion einer Eins von allen Werten der Entscheidungstabelle, in der eine "Ja"-Antwort gegeben worden war, modifiziert. Die Regeltabelle sieht jetzt folgendermaßen aus:

−1	0	−1	−1
0	−1	0	−1

Wenn Sie jetzt die ein Auto beschreibenden Werte noch einmal eingeben, wird das Programm korrekt antworten:

RÄDER J	FLÜGEL N	MOTOR J	REIFEN J
SCHIENEN N	FENSTER J	KETTEN N	STEUERBAR J

DE%=−5
IST ES EIN AUTO ? J

−1	0	−1	−1
0	−1	0	−1

Bevor Sie von sich selbst zu sehr überzeugt sind, versuchen Sie es noch einmal mit den Werten für ein Fahrrad, und das Ergebnis wird wieder falsch sein!

RÄDER J	FLÜGEL N	MOTOR N	REIFEN J
SCHIENEN N	FENSTER N	KETTEN J	STEUERBAR J

DE%=−3
IST ES EIN AUTO ? N

0	0	−1	0
0	−1	1	0

Die für das Fahrrad und das Auto übereinstimmenden Eigenschaften sind jedoch um Eins hochgezählt worden, so daß bei der nächsten Wiederholung die korrekte Entscheidung getroffen werden kann:

RÄDER J	FLÜGEL N	MOTOR N	REIFEN J
SCHIENEN N	FENSTER N	KETTEN J	STEUERBAR J

DE%=1
IST ES EIN FAHRRAD ? J

0	0	−1	0
0	−1	1	0

Die Situation hat sich nun stabilisiert, und das Programm wird nun immer in der Lage sein, bei entsprechender Eingabe zwischen einem Auto und einem Fahrrad zu unterscheiden:

RÄDER J	FLÜGEL N	MOTOR J	REIFEN J
SCHIENEN N	FENSTER J	KETTEN N	STEUERBAR J

DE%=−2
IST ES EIN AUTO ? J

0	0	−1	0
0	−1	1	0

Der endgültige Wert von DE% für ein Fahrrad ist 1 und der für ein Auto −2.

Bei Betrachtung der Regeltabellenwerte werden Sie feststellen, daß diese sowohl vom Wert als auch von der Position mit den die Objekte unterscheidenden Eigenschaften korrespondieren (Ketten für Fahrrad und Motor und Fenster für Auto).

WIR ERWEITERN UNSER SPEKTRUM

Obwohl Sie es nun geschafft haben, Ihren Computer etwas zu lehren, ist es wirklich nicht sehr weltbewegend, nur zwischen zwei Objekten unterscheiden zu können. Lassen Sie uns das Programm zur Bearbeitung eines weiteren Spektrums von

ALLE WERTE AUF NULL
MERKMALS-WERTE AUF NULL
GIB AUS MERKMAL
"Y"
JA
MERKMAL-WERT = 1
NEIN
ALLE DURCH
NEIN
JA
AKTUALISIERE ENTSCHEI-DUNGS-VARIABLEN
ALLE DURCH
NEIN
JA
DE(N) >= DE
JA
DE = DE(N)
TS = N
ALLE DURCH
NEIN
WAR ES OB$(TS) ?
"Y"
JA
NEIN
GIB AUS OBJEKT LISTE
WELCHES ?
RICHTIG ?
NO
JA
GIB AUS DE(N), DE, CR
DE(N)=DE
NEIN
JA
N <> CR
NEIN
JA
AKTUALISIERE TABELLE
ALLE DURCH
NEIN
JA
NEIN
AKTUALISIERE TABELLEN
ALLE DURCH
NEIN
GIB AUS TABELLEN

Ablaufdiagramm 6.2 Weiteres Spektrum mit Tabellen

Möglichkeiten erweitern (siehe **Ablaufdiagramm 6.2**). Am Anfang müssen wir uns die Anzahl der zu erkennenden Objekte OB% definieren, ihre Namen in DATA-Anweisungen festlegen, die wir später in eine neue Tabelle OB$(OB%) ändern, damit die Regeln für jedes Objekt getrennt abgelegt werden können, und eine Entscheidungstabelle DE(n) zum Ablegen der Entscheidungswerte für jedes Objekt erstellen.

```
10 GOSUB 10000
10000 FE%=8:OB%=5
10010 DIM FE$(FE%),FV(FE%),RU(FE%,OB%),O
B$(OB%),DE(OB%)
10020 FOR N=1 TO FE%
10030 READ FE$(N)
10040 NEXT N
10050 FOR N=1 TO OB%
10060 READ OB$(N)
10070 NEXT N
11000 DATA RAEDER,FLUEGEL,MOTOR,REIFEN,S
CHIENEN,FENSTER,KETTEN,STEUERBAR
11010 DATA FAHRRAD,AUTO,TRAIN,FLUGZEUG,P
FERD
12000 RETURN
```

Es reicht hier nicht mehr aus, mit einer einfachen Entscheidungsvariablen DE% zu arbeiten. Wir müssen den Entscheidungswert vielmehr für jedes Objekt zu jeder Zeit bestimmen. In jedem Zyklus müssen wir deshalb zuerst DE% auf Null setzen und dann jedes Element der Entscheidungstabelle DE(n) ebenfalls auf Null setzen, so daß wir für jedes Objekt mit einem "sauberen" Status starten können.

```
20 DE%=0
30 FOR N=1 TO OB%
40 DE(N)=0
50 NEXT N
```

Die Werte für jede Eigenschaft werden dann in der bekannten Art und Weise wie zuvor eingegeben.

```
60 FOR N=1 TO FE%
70 FV(N)=0
80 PRINT FE$(N);"  ";
```

```
90 GET IN$:IF IN$="" THEN 90
100 PRINT IN$
110 IF IN$="J" THEN FV(N)=1
120 NEXT N
```

Jedes Element der Entscheidungstabelle DE(n) wird nun entsprechend des Status' der eingegebenen Werte FV(n) und des Inhalts des entsprechenden Regeltabellenelementes RU(n,m) geändert.

```
130 FOR N=1 TO FE%
140 FOR M=1 TO OB%
150 DE(M)=DE(M)+FV(N)*RU(N,M)
160 NEXT M,N
```

Jetzt brauchen wir nur noch zu überprüfen, ob irgendeiner der Entscheidungswerte für irgendeines der Objekte DE(n) größer oder gleich dem Entscheidungswert DE% ist. In diesem Fall setzen wir eine "Höchstpunkt"-Variable TS% gleich der Zahl des Objektes, das diese Entscheidung verursacht hat.

```
170 FOR N=1 TO OB%
180 IF DE(N)>=DE% THEN DE%=DE(N):TS%=N
190 NEXT N
```

Das System nimmt nun an, daß dies die korrekte Antwort ist, fragt wieder um Bestätigung und kehrt ohne jede Änderung zum Programmanfang zurück, wenn die Antwort korrekt war.

```
200 PRINT "KANN ES EIN ";OB$(TS%);" SEIN
 ";
210 GET IN$:IF IN$="" THEN 210
215 PRINT IN$
220 IF IN$="J" THEN 20
```

War die Antwort nicht korrekt, werden die Namen und Nummern aller Objekte ausgedruckt, und Sie werden nach der Nummer der korrekten Antwort CR% gefragt. Um den Abbruch des Programms durch die Eingabe eines nicht erlaubten Wertes für CR% zu verhindern, wird darauf im Programm besonders abgefragt.

```
230 FOR N=1 TO OB%
240 PRINT N,OB$(N)
250 NEXT N
260 PRINT "WAS WAR ES";
270 GET CR%:IFCR%<1 OR CR%>5 THEN 270
275 PRINT CR%
```

Danach wird abgeprüft, ob der Entscheidungswert für jedes Objekt DE(n) größer oder gleich dem allgemeinen Entscheidungswert DE% ist, *und* ob das in Betracht gezogene Objekt *nicht* die korrekte Antwort ist. Wenn diese *beiden* Bedingungen zutreffen, werden die Regeln wieder durch Subtraktion der Werte für die korrekten Eigenschaften FV(n) zum Vorteil der korrekten Antwort korrigiert.

```
280 FOR N=1 TO OB%
290 PRINT DE(N),DE%,CR%
300 IF DE(N)>=DE% AND N<>CR% THEN FOR M=
1 TO FE%:RU(M,N)=RU(M,N)-FV(M):NEXT M
310 NEXT N
```

Danach werden die Werte der korrekten Eigenschaften FV(n) zu denen der korrekten Objekte der Regeltabelle addiert.

```
320 FOR M=1 TO FE%
330 RU(M,CR%)=RU(M,CR%)+FV(M)
340 NEXT M
```

Zum Abschluß wird der Status der Regeltabelle ausgedruckt, so daß Sie sehen können, welche Änderungen sich ergeben haben.

```
350 FOR M=1 TO OB%
360 FOR N=1 TO FE%
370 PRINT RU(N,M);
380 NEXT N
390 PRINT
400 NEXT M
410 GOTO 20
```

Auch in diesem Fall erklärt ein Beispiel mehr als viele Wörter. Überprüfen Sie bitte selbst, was bei der Eingabe der nächsten Folge passiert:

RÄDER J	FLÜGEL N	MOTOR N	REIFEN J
SCHIENEN N	FENSTER N	KETTEN J	STEUERBAR J

Das Programm wird mit der fehlerhaften Annahme, daß es sich um ein Pferd handelt, zurückkommen. Sie müssen ihm nun sagen, daß diese Annahme fehlerhaft ist und es darauf hinweisen, was die korrekte Antwort gewesen wäre:

KANN ES EIN PFERD SEIN

1 FAHRRAD
2 AUTO
3 EISENBAHN
4 FLUGZEUG
5 PFERD

WAS WAR ES 1

Die Stati der verschiedenen Entscheidungs- und Regeltabellen werden nun zu Ihrer Information ausgedruckt. (Beachten Sie bitte, daß die hier zu Ihrer Information angegebenen Variablen-Namen auf dem Bildschirm nicht erscheinen.)

DE(N)	DE%	CR%
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1

1	0	0	1	0	0	1	1	(Fahrrad)
−1	0	0	−1	0	0	−1	−1	(Auto)
−1	0	0	−1	0	0	−1	−1	(Eisenbahn)
−1	0	0	−1	0	0	−1	−1	(Flugzeug)
−1	0	0	−1	0	0	−1	−1	(Pferd)
A	B	C	D	E	F	G	H	

(A=Räder	B=Flügel	C=Motor	D=Reifen
E=Schienen	F=Fenster	G=Ketten	H=steuerbar)

Bei genauer Betrachtung werden Sie feststellen, daß einige Eigenschaften Änderungen in den Regeltabellen verursacht haben. Es handelt sich hier um die Eigenschaften "Räder", "Reifen", "Ketten" und "steuerbar" – alles Eigenschaften, die wir als die eines Fahrrades definiert haben, und die es natürlich nicht bei einem Pferd gibt. Sie werden auch feststellen, daß die Werte dieser Eigenschaften in der Fahrradzeile nun alle plus eins sind, während die Werte dieser Eigenschaften für alle anderen Objekte minus eins sind.

Als nächstes wollen wir die Eigenschaften eines Autos eingeben, von dem der Computer annimmt, daß es ein Fahrrad ist und ihn dann entsprechend korrigieren. Beachten Sie, daß die Regeltabellen für Fahrrad und Auto entsprechend der neuen Information geändert werden.

```
RÄDER J          FLÜGEL N       MOTOR J        REIFEN J
SCHIENEN N       FENSTER J      KETTEN N       STEUERBAR J

KANN ES EIN FAHRRAD SEIN N

1    FAHRRAD
2    AUTO
3    EISENBAHN
4    FLUGZEUG
5    PFERD

WAS WAR ES 2

 3          3          2
-3          3          2
-3          3          2
-3          3          2
-3          3          2

 0    0   -1    0    0   -1    1     0     (Fahrrad)
 0    0    1    0    0    1   -1     0     (Auto)
-1    0    0   -1    0    0   -1    -1     (Eisenbahn)
-1    0    0   -1    0    0   -1    -1     (Flugzeug)
-1    0    0   -1    0    0   -1    -1     (Pferd)

 A    B    C    D    E    F    G     H
```

(A=Räder B=Flügel C=Motor D=Reifen
E=Schienen F=Fenster G=Ketten H=steuerbar)

Beim nächsten Mal definieren wir ein Flugzeug, und das Programm entscheidet sich für ein Auto, wonach wir wieder korrigieren.

RÄDER J	FLÜGEL J	MOTOR J	REIFEN J
SCHIENEN N	FENSTER J	KETTEN N	STEUERBAR J

KANN ES EIN AUTO SEIN N

1 FAHRRAD
2 AUTO
3 EISENBAHN
4 FLUGZEUG
5 PFERD

WAS WAR ES 4

Und nun versuchen wir es mit einer Eisenbahn, die es immer noch nicht richtig findet!

RÄDER J	FLÜGEL N	MOTOR J	REIFEN N
SCHIENEN N	FENSTER J	KETTEN N	STEUERBAR N

KANN ES EIN FLUGZEUG SEIN N

1 FAHRRAD
2 AUTO
3 EISENBAHN
4 FLUGZEUG
5 PFERD

WAS WAR ES 3

Und am Ende ein Pferd, das als Flugzeug erkannt wird!

RÄDER N	FLÜGEL N	MOTOR N	REIFEN N
SCHIENEN N	FENSTER N	KETTEN N	STEUERBAR J

KANN ES EIN FLUGZEUG SEIN N

1 FAHRRAD
2 AUTO

3 EISENBAHN
4 FLUGZEUG
5 PFERD

WAS WAR ES 5

Wenn Sie Ihren Experten weiterhin mit Informationen füttern, wird er am Ende in der Lage sein, jedesmal die richtige Antwort zu geben. Wie lange dies dauert, hängt davon ab, wie unterschiedlich die einzelnen Objekte sind, und in welcher Reihenfolge die Objekte dem Expertensystem präsentiert werden.
Bei unglücklichen Kombinationen kann es sehr lange dauern, bis das Expertensystem unfehlbar wird. Mit der folgenden Sequenz hat das Programm alle Regeln gelernt und war danach in der Lage, jede Entscheidung richtig zu treffen.

Flugzeug (Eisenbahn)	Auto (Flugzeug)	Fahrrad (JA)
Auto (JA)	Flugzeug (Auto)	Flugzeug (JA)
Pferd (JA)	Flugzeug (Fahrrad)	Auto (Flugzeug)
Flugzeug (Auto)	Flugzeug (Auto)	Auto (Flugzeug)
Auto (JA)	Flugzeug (Auto)	Flugzeug (JA)
Auto (JA)	Flugzeug (JA)	Pferd (JA)
Fahrrad (JA)	Eisenbahn (Auto)	Eisenbahn (JA)
Fahrrad (JA)	Auto (Flugzeug)	Auto (JA)
Flugzeug (Auto)	Flugzeug (JA)	Auto (Flugzeug)
Auto (JA)	Flugzeug (JA)	Auto (JA)
Fahrrad (Auto)	Auto (JA)	Flugzeug (JA)
Eisenbahn (JA)	Pferd (JA)	Fahrrad (JA)

Wenn Sie den endgültigen Status der Regeltabelle nach diesem Stand sehen wollen, stoppen Sie das Programm und geben Sie GOTO 350 als direktes Kommando ein. Wie Sie sehen, reicht die Skala der Werte von +6 bis −2. Es ist also nicht erstaunlich, daß es so lange gedauert hat, bis das Programm diesen Status erreicht hat.

1	0	−1	1	0	−2	3	0	(Fahrrad)
−1	4	1	0	−1	1	−2	0	(Auto)
0	−1	1	−2	2	1	−1	−2	(Eisenbahn)
−2	6	0	0	−1	0	−2	−2	(Flugzeug)
−1	0	0	−1	0	0	−1	0	(Pferd)
A	B	C	D	E	F	G	H	

(A=Räder	B=Flügel	C=Motor	D=Reifen
E=Schienen	F=Fenster	G=Kette	H=steuerbar)

In der Praxis wird man natürlich solch ein Expertensystem nicht im Rahmen eines Dialoges mit Informationen versehen, sondern ihm bereits vorher gesammelte Informationen und Entscheidungshilfen aus einem bestimmten Bereich übergeben, um es dann selbst daran arbeiten zu lassen. Da diese Wege in Tabellen abgespeichert werden, können Sie sie leicht mit einer kleinen Routine für einen späteren Gebrauch sichern.

KAPITEL 7

UNGEFÄHRE ÜBEREINSTIMMUNGEN

Computer verhalten sich vollkommen logisch. Im Gegensatz dazu sind unsere Speicherbänke sehr unzuverlässig, was immer wieder zu Problemen führen kann, wenn wir Informationen über ein bestimmtes Subjekt wiederfinden wollen. Z. B. passiert es oft, daß "gleiche" (oder ähnliche) Familiennamen völlig unterschiedlich geschrieben werden.
Um diesen Problemen der unterschiedlichen Schreibweise begegnen zu können, hat man bereits im Jahre 1890 für die Volkszählung in den Vereinigten Staaten von Amerika ein Hilfsmittel entwickelt. Man versuchte dort, Übereinstimmungen aufgrund des Wortklanges zu erkennen. Bei entsprechender Codierung konnte man erreichen, daß ähnlich klingende Wörter auch eine ähnliche Codefolge hatten.
Für die Codierung einzelner Wörter legte man folgende Regeln fest:

1. Der erste Buchstabe des Wortes ergibt immer das erste Zeichen des Codes.

Für den zweiten und alle folgenden Buchstaben galten folgende Regeln:

2. Vokale werden ignoriert.

3. Die Buchstaben W, Y, Q und H werden ignoriert.

4. Satzzeichen werden ignoriert.

5. Die anderen Buchstaben werden mit den Werten 1 bis 6 wie folgt codiert:

Buchstaben	Code
bfpv	1
cgjksxz	2
dt	3
i	4
mn	5
r	6

6. Wenn der Folgebuchstaben den gleichen Code hat, wird nur der Code des ersten weiterverwendet.

7. Es werden maximal die ersten vier Zeichen des Codes verwendet.

8. Wenn der Code kürzer ist als vier Zeichen, wird er mit Nullen auf vier Zeichen aufgefüllt.

Hierzu einige Beispiele:

HAUS – H200

(H bleibt erhalten, A und U fallen weg, S ergibt 2)

BANDAGE – B532

(B bleibt erhalten, A fällt weg, N gibt 5, D gibt 3, A fällt weg, G gibt 2)

IRLAND – I645

(I bleibt erhalten, R gibt 6, L gibt 4, A fällt weg, N gibt 5)

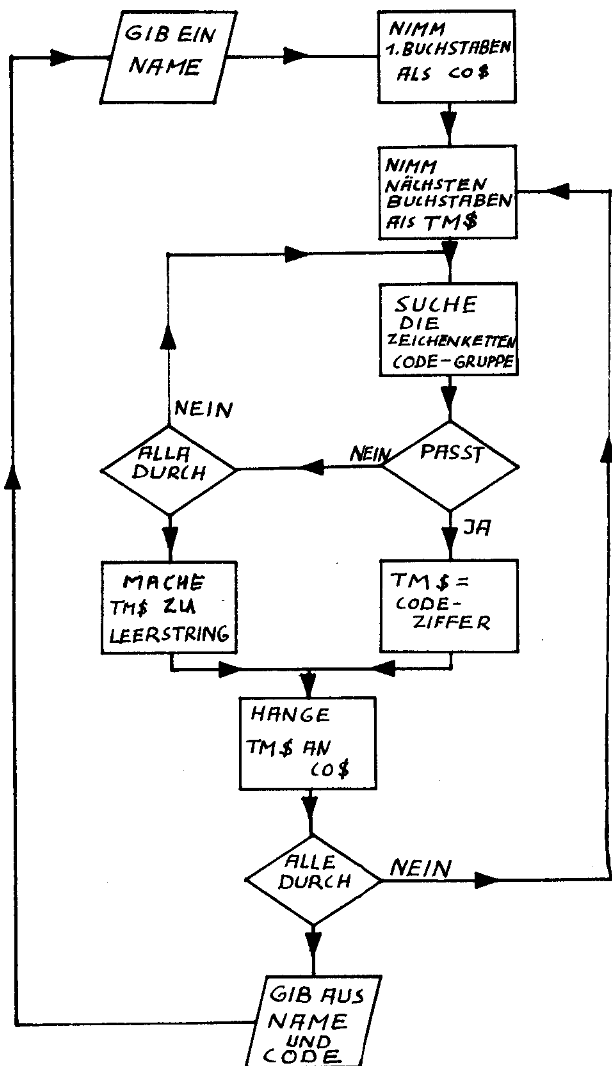
CODIERUNGSPROGRAMM

Um die bis jetzt investierte geistige Arbeit auch in der Zukunft verwenden zu können, lassen Sie uns ein Programm entwickeln, das die Eingabe eines Wortes in Deutsch und die Ausgabe im "Klangcode" erlaubt (siehe **Ablaufdiagramm 7.1**). Als erstes springen wir in ein Unterprogramm, das alle zur Berechnung des Codes verwendeten Buchstaben gruppenweise in je ein Element der Klangcodetabelle SC$(n) einliest. Dabei sind die Buchstaben entsprechend ihres Codewertes sortiert.

```
10 GOSUB 10000
10000 DIM SC$(6)
10010 DATA BFPV,CGJKSXZ,DT,L,MN,R
12000 FOR N=1 TO 6
12010 READ  SC$(N)
12020 NEXT N
13000 RETURN
```

Nun können wir das zu codierende Wort IN$ eingeben und beginnen mit der Codierung des Ergebnisses CO$ mit dem ersten Buchstaben des Wortes.

```
100 INPUT IN$
110 CO$=LEFT$(IN$,1)
```



Ablaufdiagramm 7.1 Mache einen Klangcode

Danach untersuchen wir die restlichen Buchstaben des Wortes, 2 TO LEN(IN$). Dabei legen wir jeden einzelnen Buchstaben zeitweise in der Zeichenkette TM$ ab.

```
20 FOR N=2 TO LEN(IN$)
30 TM$=MID$(IN$,N,1)
```

Die wirkliche Codierung des Buchstabens in den Code programmieren wir als Unterprogramm in Zeile 1000, das wir dann jeweils aufrufen können.

```
140 GOSUB 1000
```

Jetzt müssen wir die Zeichenkette TM$ gegen jeden einzelnen Buchstaben jeder Buchstabengruppe SC$(n) prüfen. Um jede Buchstabengruppe abzuprüfen, müssen wir die Schleife sechsmal durchlaufen, den laufenden Klangcode in einen Suchbereich SE$ ablegen und in die ISTR-Routine springen, die jeden Buchstaben der Gruppe gegen TM$ abprüft.

```
1000 FOR P=1 TO 6
1010 SE$=SC$(P)
1020 GOSUB 5000
```

Die INSTR-Routine ist der ähnlich, die wir bereits in den vorherigen Kapiteln benutzt haben.

Nachdem die INSTR-Routine durchgeführt ist, müssen wir feststellen, ob bzw. welche Klanggruppe gefunden wurde. Bei keiner Übereinstimmung wird SP% auf Null gesetzt. Bei einer Übereinstimmung wird SP% auf M gesetzt und zeigt damit auf den Wert der gefundenen Codegruppe.

```
5000 FOR M=1 TO LEN(SE$)
5010 IF MID$(SE$,M,1)=TM$ THEN SP%=M:RET
URN
5020 NEXT M
5030 SP%=0
5040 RETURN
```

Bei einer Übereinstimmung weisen wir der Zeichenkette TM$ den "zeichenmäßigen" Inhalt der Laufvariable P zu. (Das STR$-Kommando konvertiert eine Zahl in eine Zeichenkette, aber wir müssen mit RIGHT$ noch das dadurch am Anfang zugefügte Leerzeichen ausblenden.)

```
1030 IF SP%>0 THEN TM$=RIGHT$(STR$(P),1)
:RETURN
```

Wenn in dieser Gruppe keine Übereinstimmung gefunden wurde, müssen wir die nächste Gruppe überprüfen.

```
1040 NEXT P
```

Wenn in keiner der Gruppen eine Übereinstimmung gefunden wurde, enthält TM$ eines der zu ignorierenden Zeichen. Wir löschen den Inhalt von TM$ und springen in das Hauptprogramm zurück.

```
1050 TM$=""
1060 RETURN
```

Unser Klangcode CO$ ergibt sich nun aus dem ersten übernommenen Zeichen und dem zuletzt codierten Zeichen TM$.

```
170 CO$=CO$+TM$
```

Wir springen nun in unsere Schleife zurück, um das nächste Zeichen in IN$ zu behandeln.

```
180 NEXT N
```

Nachdem wir IN$ vollständig untersucht haben, drucken wir die Eingabe IN$ und den gesamten Klangcode CO$ aus, bevor wir zur Zeile 100 zur Anforderung einer neuen Eingabe zurückspringen.

```
210 PRINT:PRINT "NAME","CODE":PRINT IN$,
CO$
320 GOTO 100
```

Bei der Benutzung dieser Routine werden jedoch die Codes noch nicht mit Nullen aufgefüllt bzw. auf vier Zeichen verkürzt, und gleiche Codes werden noch wiederholt.

WIR KÜMMERN UNS UM DIE DETAILS

Um das Problem der Wiederholung des gleichen Codes für benachbarte Buchstaben in den Griff zu bekommen, merken wir uns die letzte Zeichenkette LT$. Am Anfang weisen wir ihr das erste Zeichen von IN$ zu, so daß dieses Zeichen nicht mehr wiederholt wird. Wenn wir nun unsere FOR-NEXT-Schleife bearbeiten, müssen wir nur noch LT$ mit TM$ vergleichen. Bei Gleichheit dürfen wir TM$ nicht zu unserem Code CO$ addieren. Im anderen Fall erhält LT$ den Wert des letzten TM$.

```
110 TM$=LEFT$(IN$,1):CO$=TM$:GOSUB 1000:
LT$=TM$
 150 IF TM$=LT$ THEN GOTO 180
 160 LT$=TM$
```

Nun können wir uns um das Problem der zu kurzen Codes kümmern. Als erstes überprüfen wir, ob unser Code CO$ kürzer als vier Zeichen ist. Ist dies der Fall, fügen wir drei Nullen an das Ende an und reduzieren unseren Code mit LEFT$ wieder auf vier Zeichen.

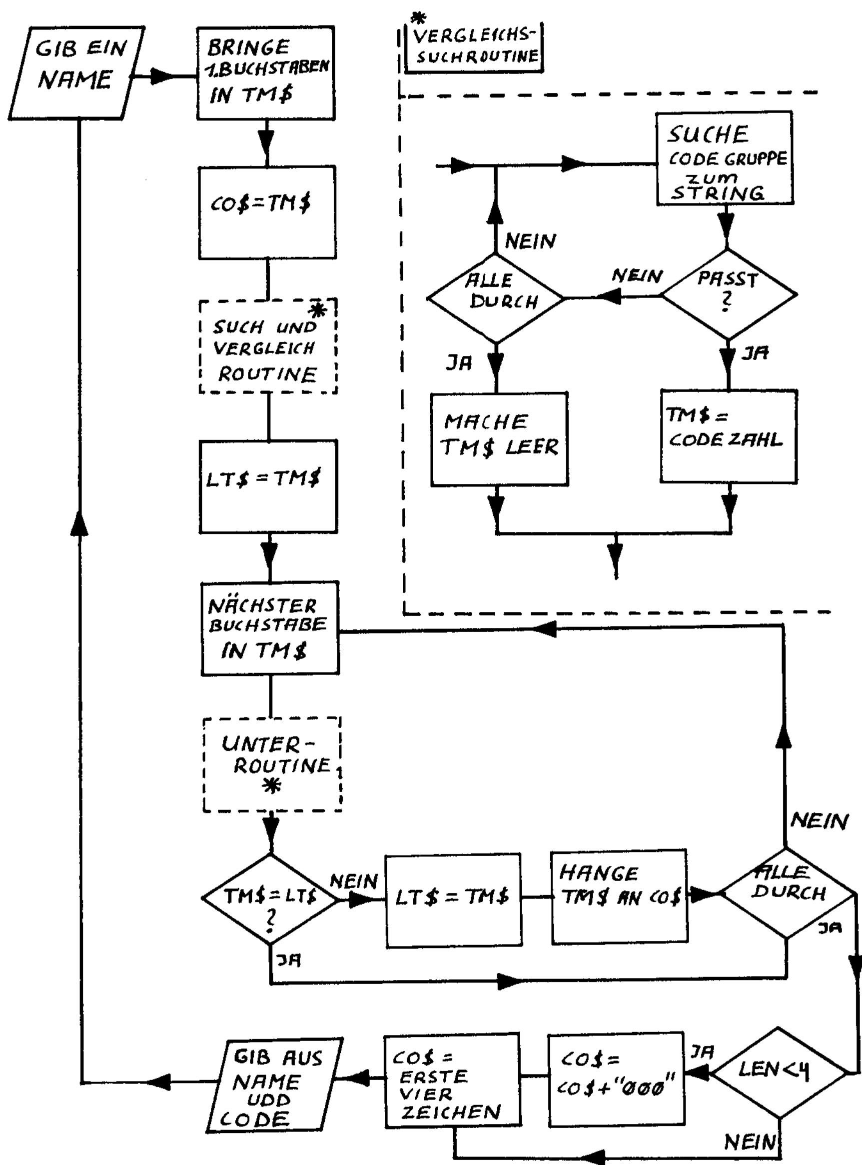
```
190 IF LEN(CO$)<4 THEN CO$=CO$+"000":CO$
=LEFT$(CO$,4)
```

War unser Code von Anfang an zu lang, wird er ohne vorherige Verlängerung auf vier Zeichen verkürzt.

```
200 IF LEN(CO$)>4 THEN CO$=LEFT$(CO$,4)
```

WIR SUCHEN ÜBEREINSTIMMUNGEN

Nachdem wir nun eine zuverlässige Methode zum Berechnen des Klangcodes haben, wollen wir sie auch ausprobieren. Als erstes lesen wir eine Gruppe von Namen aus DATA-Anweisungen in eine Namentabelle NA$(n). In unserem Beispiel benutzen wir nur 18 Namen, wenn Sie mehr benutzen wollen, hilft Ihnen bestimmt ein kurzer Blick in das Telefonbuch. Die Anzahl der Wörter legen wir in der Variablen NW% ab.



Ablaufdiagramm 7.2 Detaillierter Ablauf

```
10010 NW%=17:DIM NA$(NW%)
11010 DATA ABRAHAM,ABRAHAMS,ABRAMS,ADAM,
ADAMS,ADDAMS,ADAMSON,ALAN,ALLAN,ALLEN
11020 DATA ANTHANY,ANTHONY,ANTON,ANTROPU
S,APFELBAUM,APPEL,ARPEL,AZTEKAN
12030 FOR N=0 TO 17
12040 READ NA$(N)
12050 NEXT N
```

Der Sinn der Benutzung des Klangcodes liegt darin, daß Sie mit Hilfe des Klangcodes eine Übereinstimmung herausfinden, bevor Sie die möglichen Wörter ausdrukken. Dazu müssen wir die Codes aller Namen, die wir aus den DATA-Anweisungen eingelesen haben, bestimmen und in eine Tabelle NC$(n) ablegen. Die Routine zum Finden des Klangcodes ist im Prinzip die gleiche, die wir bereits für das Auffinden des Codes der Eingabe IN$ benutzt haben.

```
10020 DIM NC$(NW%)
12060 PRINT:PRINT "NAME","CODE":PRINT
12070 FOR Q=0 TO NW%
12080 PRINT NA$(Q),
12090 TM$=LEFT$(NA$(Q),1):CO$=TM$:GOSUB
1000:LT$=TM$
12100 FOR N=2 TO LEN(NA$(Q))
12110 TM$=MID$(NA$(Q),N,1)
12120 GOSUB 1000
12130 IF TM$=LT$ THEN NEXT N:GOTO 12170
12140 LT$=TM$
12150 CO$=CO$+TM$
12160 NEXT N
12170 IF LEN(CO$)<4 THEN CO$=CO$+"000":C
O$=LEFT$(CO$,4)
12180 IF LEN(CO$)>4 THEN CO$=LEFT$(CO$,4
)
12190 PRINT CO$
12200 NC$(Q)=CO$
12210 NEXT Q
```

Wenn Sie das Programm nun laufen lassen, werden vor der Anforderung einer Eingabe alle in DATA-Anweisungen abgelegten Namen und deren Codes ausgedruckt.

ABRAHAM	A165
ABRAHAMS	A165
ABRAMS	A165
ADAM	A350
ADAMS	A352
ADDAMS	A352
ADAMSON	A352
ALAN	A450
ALLAN	A450
ALLEN	A450
ANTHANY	A535
ANTHONY	A535
ANTON	A535
ANTROPUS	A536
APFELBAUM	A141
APPEL	A140
ARPEL	A614
AZTEKAN	A232

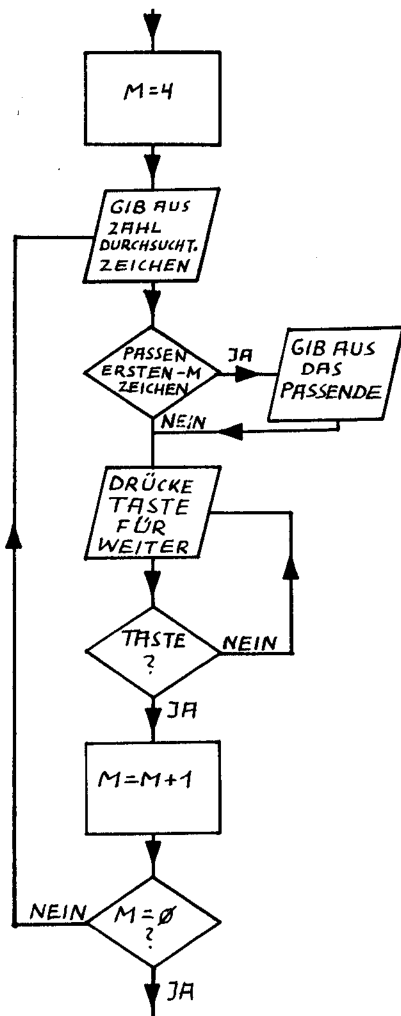
Wir müssen jetzt nur noch herausfinden, welche dieser Codes mit dem Code Ihrer Eingabe übereinstimmen und diese Namen dann mit einer FOR-NEXT-Schleife ausdrucken.

```
240 PRINT
250 FOR N=0 TO NW%
260 IF CO$=NC$(N) THEN PRINT NA$(N),NC$(
N)
270 NEXT N
```

Dieses Programm druckt nur Wörter aus, die exakt mit dem entsprechenden Klangcode übereinstimmen. Wenn Sie einen anderen Namen eingeben, z. B. DARMSTADT, werden Sie folgende Antwort bekommen:

DARMSTADT D652

Obwohl DARMSTADT nicht in den DATA-Anweisungen vorhanden ist, können wir sowohl DARMSTADT wie auch andere finden.



Ablaufdiagramm 7.3 Teilvergleiche

WIR ÜBERPRÜFEN AUF TEILWEISE ÜBEREINSTIMMUNG

Wie Sie oben sehen, wurde auch DARMSTADT in einen Klangcode umgewandelt. Es würde uns deshalb helfen, wenn wir auch teilweise Übereinstimmungen ausdrucken könnten.
Dies könnten wir sehr einfach durch das Hinzufügen einer zusätzlichen FOR-NEXT-Schleife erreichen, die jeweils einen kleiner werdenden Teil der Eingabe LEFT$(CO$,M) mit kleiner werdenden Abschnitten der gespeicherten Codes LEFT$(NC$),M) vergleicht (siehe **Ablaufdiagramm 7.3**). Die FOR-NEXT-Schleife wird in Zeile 260 angesprochen.

```
230 FORM=4 TO 1 STEP-1
240 PRINT:PRINT M;"ZEICHENUEBEREINSTIMMU
NG":PRINT
260 GOSUB 14000
280 PRINT:PRINT "DRUECK EINE TASTE FUER
WEITER"
290 GET I$:IF I$="" THEN 290
300 PRINT:PRINT
310 NEXT M

14000 FORI=1TONW%
14010 IFLEFT$(CO$,M)=LEFT$(NC$(I),M) THE
N PRINT NA$(I),N$(N)
14020 NEXT
14030 RETURN
```

Wenn Sie nun ANTHANAS versuchen, wird Ihnen der volle Bereich der Möglichkeiten ausgegeben.

```
NAME            CODE
ANTHANAS        A535

4 Zeichenübereinstimmung
ANTHANY         A535
ANTHONY         A535
ANTON           A535
FORTSETZUNG ? DANN EINE TASTE DRUECKEN
```

3 Zeichenübereinstimmung

```
ANTHANY        A535
ANTHONY        A535
ANTON          A535
ANTROPUS       A536
FORTSETZUNG ? DANN EINE TASTE DRUECKEN
```

2 Zeichenübereinstimmung

```
ANTHANY        A535
ANTHONY        A535
ANTON          A535
ANTROPUS       A536
FORTSETZUNG ? DANN EINE TASTE DRUECKEN
```

1 Zeichenübereinstimmung

```
ABRAHAM        A165
ABRAHAMS       A165
ABRAMS         A165
ADAM           A350
ADAMS          A352
ADDAMS         A352
ADAMSON        A352
ALAN           A450
ALLAN          A450
ALLEN          A450
ANTHANY        A450
ANTHONY        A535
ANTON          A535
ANTROPUS       A536
APFELBAUM      A141
APPEL          A140
ARPEL          A614
AZTEKAN        A232
FORTSETZUNG ? DANN EINE TASTE DRUECKEN
```

KAPITEL 8

WIR ERKENNEN UMRISSE

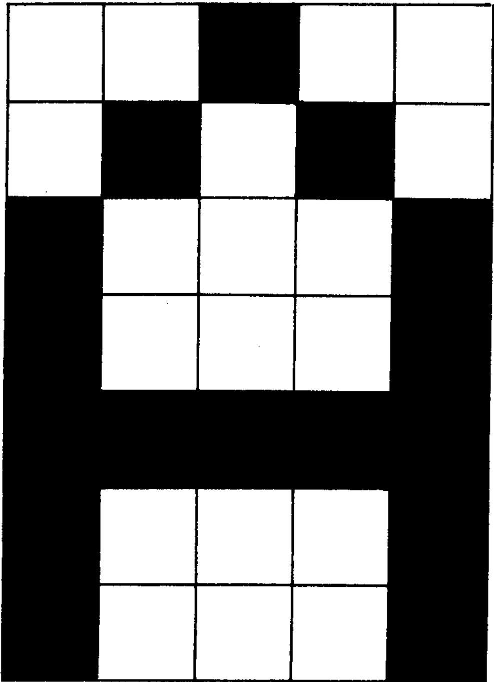
Normalerweise erkennen wir Objekte unter Benutzung unserer Sinne. Wir können sehen, hören, tasten und fühlen. Unser Computer kann natürlich nur Informationen über die Tastatur bekommen. Um Ihrem Computer einen genaueren Eindruck von der ihn umgebenden Welt zu geben, müßten wir mit einem erheblichen Einsatz an Wissen und Erfahrung elektronische oder mechanische Sensoren anschließen. Wir werden uns statt dessen mit der Simulation von Lichtsensoren begnügen, um das Erkennen von Umrissen zu demonstrieren.
Am Anfang wollen wir versuchen, drei einfache Umrisse zu erkennen: Eine vertikale Linie, eine Quadrat und ein rechtwinkliges Dreieck.
Wir können uns diese Umrisse in einem imaginären Gitternetz (8 x 8) vorstellen und dann entscheiden, ob ein Punkt bei jeder Koordinate vorhanden ist oder nicht.
Im Falle der Linie ist nur die erste X-Koordinate, aber alle Y-Koordinaten benutzt. Ein Quadrat ist ein bißchen komplizierter, da alle X-Koordinaten der Y-Reihen 1 und 8 gesetzt sind und von den Y-Reihen 2 und 7 nur der erste und letzte X-Punkt gesetzt sind. Das Dreieck ist sogar noch komplizierter, da der Linienzug durch das jeweilige Hochzählen der X-Koordinate errreicht wird.

Y-Zeile	Linie	Quadrat	Dreieck
1	1	255	1
2	1	129	3
3	1	129	5
4	1	129	9
5	1	129	17
6	1	129	33
7	1	129	65
8	1	255	255

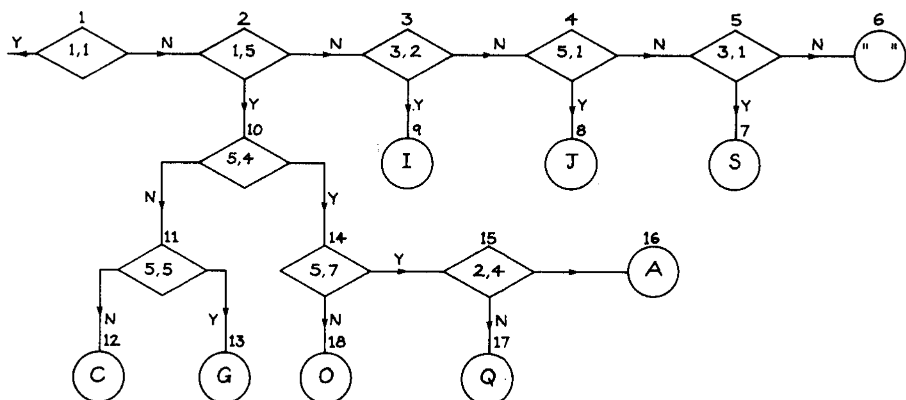
Tabelle 8.1 Dezimalwerte der Umrisse in binärer Form

Ein Weg, diese speziellen Figuren zu beschreiben, wäre die Darstellung jedes Punktes durch ein einfaches Bit und die Berechnung eines Dezimalwertes für jede Zeile, wie wir es bei unseren Expertensystemen bereits verwendet haben (siehe **Tabelle 8.1**). Mit diesem Verfahren werden die Zeichen dargestellt, die Sie auf Ihrem Bildschirm sehen. Die Formate dafür sind im Speicher in dieser Art abgelegt. Das Bild 8.1 zeigt z. B., wie der Buchstabe "A" aufgebaut ist.
Es gibt aber Maschinen, sogenannte "Klarschriftleser" die diesen Prozeß in umge-

kehrter Reihenfolge beherrschen. Sie sind in der Lage, gedruckte Informationen einem Gittermuster entsprechend abzuprüfen und festzustellen, ob bei der jeweiligen Position Licht reflektiert wird oder nicht.



8.1 Aufbau des Buchstabens "A"



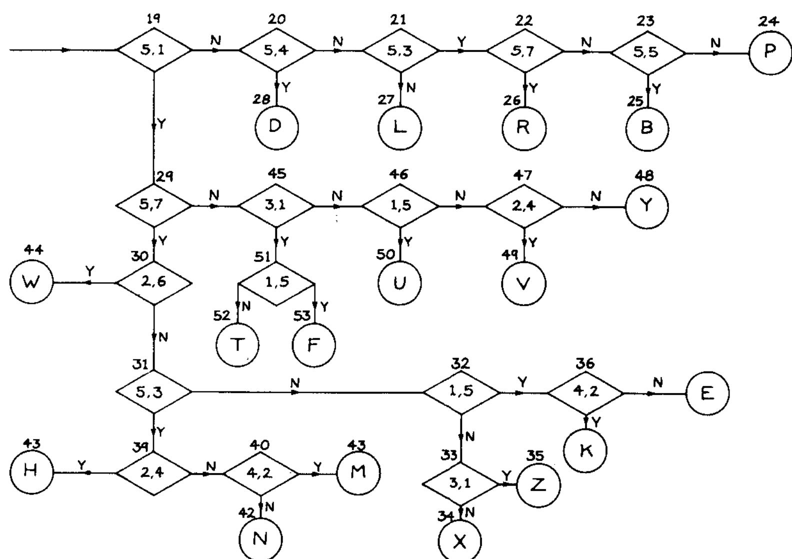


Abbildung 8.2 Entscheidungsbaum für das Alphabet

X

Y	1	2	3	4	5
1	X		X		X
2			X	X	
3					X
4		X			X
5	X				X
6		X			
7					X

Abbildung 8.3 Punkte im Entscheidungsbaum

Sie lesen in Wirklichkeit ein "Ja"- und- "Nein"-Muster für jede Koordinate, die diese Information codieren und sie mit den Mustern bekannter Umrisse vergleichen. Die einfachste Art eines Vergleichs wäre, jeden Punkt als ein Bit aufzufassen, jede Zeile in einen Dezimalwert umzurechnen und diesen dann mit der Tabelle der bekannten Werte zu vergleichen. Dies hat jedoch den großen Nachteil, daß wir jeden einzelnen Punkt eines möglichen Gitters überprüfen müssen.

WIR OPTIMIEREN UNSERE ENTSCHEIDUNGEN

Eine schnellere Methode beruht auf der Tatsache, daß jedes Zeichen aufgrund der Überprüfung einiger kritischer Eigenschaften erkannt werden kann. In **Bild 8.2** ist z. B. der Entscheidungsbaum für alle Großbuchstaben des Alphabets unter Benutzung von nur 12 Punkten (siehe **Bild 8.3**) angegeben. In einigen Fällen ist es sogar noch nicht einmal nötig, alle 12 Punkte zu überprüfen. Bei der Verfolgung der Entscheidungspfade werden Sie feststellen, daß mit sieben Schritten jeder Buchstabe gefunden werden kann, und daß die meisten Buchstaben sogar schon mit weniger als fünf Schritten gefunden werden. Dies ist auf jeden Fall schneller, als die Überprüfung von 8 x 8 = 64 Einzelpunkten.

3 Schritte – I, D
4 Schritte – L, J, C, G, O, W
5 Schritte – S, A, Q, R, T, F, U, Leerzeichen
6 Schritte – P, V, Y, H
7 Schritte – B, M, N, E, K, X, Z

Um Ihnen dieses Verfahren zu demonstrieren, simulieren wird die Arbeitsweise des Prüfkopfes, in dem wir ein Gitter auf dem Bildschirm darstellen, indem Sie Ihre Zeichen konstruieren können.
Die Bildschirmstartadresse 1024 und das FarbRAM-Offset 54272 sind als Variable TS bzw. CO definiert. Der Bildschirm wird gelöscht, und ein dunkler Bereich von 6 x 8 Blöcken wird in der linken oberen Ecke angelegt. Zum Markieren des aktuellen Arbeitsbereiches wird ein hellerer Bereich von 5 x 7 Blöcken in den größeren Bereich gelegt.

```
10 GOSUB 10000
10000 TS=1024:CO=54272:REM SS=1024+481
12000 PRINT "♥"
12010 FOR N=1 TO 10
12020 PRINT
```

```
12030 NEXT N
13000 FOR X=0 TO 6
13010 FOR Y=0 TO 8
13020 POKE TS+CO+X+(Y*40),11
13030 POKE TS+X+(Y*40),224
13040 NEXT Y,X
13050 FOR X=1 TO 5
13060 FOR Y=1 TO 7
13070 POKE TS+CO+X+(Y*40),1
13080 NEXT Y,X
13090 X=1:Y=1
13100 RETURN
```

Mit einem blinkenden Cursor wird Ihre Position angezeigt. CP ist die laufende Position auf dem Bildschirm, TS + Offset (der jeweiligen Farbe) wird durch PEEKen der entsprechenden Position im FarbRAM nach CC gebracht. Eine andere Farbe (CC + 4) wird dann an diese Stelle gePOKEt und die Originalfarbe (CC) zurückgePOKEt, damit der alte Zustand wiederhergestellt wird.

```
20 GET A$
30 CP=TS+X+(Y*40):CC=PEEK(CP+CO):POKE CP
+CO,CC+4:POKE CP+CO,CC
40 IF A$="" THEN 20
```

Die X- und Y-Koordinaten werden entsprechend der Bewegung des Cursors geändert, und beim Drücken der Leerzeichentaste wird die Farbe der entsprechenden Position auf Schwarz (0) gesetzt. Wenn Ihnen ein Fehler unterläuft, können Sie durch Drücken der Rücktaste die Farbe der laufenden Position wieder auf 1 setzen oder mit CLR zur Aufbauroutine in Zeile 13000 zurückspringen und damit das jeweilige Gitter löschen. Durch das Drücken der RETURN-Taste erreichen Sie die Decodierungsroutine. Wenn keine dieser Tasten gedrückt wurde, wird auf erneuten Tastendruck abgeprüft.

```
50 IF A$="■" THEN X=X+1
60 IF A$="■" THEN X=X-1
70 IF A$="■" THEN Y=Y+1
80 IF A$="□" THEN Y=Y-1
90 IF A$=" " THEN POKE TS+CO++X+(Y*40),0

100 IF A$="■" THEN POKE TS+CO+X+(Y*40),1
```

```
110 IF A$="▄" THEN GOSUB 13000
120 IFASC(A$)=13 THEN 2000
170 GOTO 20
```

Um den Cursor am Verlassen unseres 5 x 7 Gitters zu hindern, müssen entsprechende Grenzen gesetzt werden.

```
130 IF X<1 THEN X=1
140 IF X>5 THEN X=5
150 IF Y<1 THEN Y=1
160 IF Y>7 THEN Y=7
```

Der Entscheidungsbaum ist in einer Reihe von verknüpften Tabellen abgelegt, wobei NB die Anzahl der Entscheidungspunkte ist, LE$(n) die Namen der Buchstaben enthält, C1(n) und C2(n) die X- bzw. Y-Koordinaten für die nächste Überprüfung enthalten, und N(n) bzw. J(n) das nächste zu benutzende Element für den "Nein"- bzw. "Ja"-Fall enthalten.

```
11000 NB=53
11010 DIM LE$(NB),C1(NB),C2(NB),N(NB),Y(
NB)
11020 FOR N=1 TO NB
11030 READ LE$(N),C1(N),C2(N),N(N),Y(N)
11040 NEXT N
```

Die DATA-Anweisungen gibt man am besten als 53 getrennte Zeilen (je eine für jeden Entscheidungspunkt) ein, da man dann eventuelle Fehler leicht ändern kann.

```
14010 DATA ,1,1,2,19
14020 DATA ,1,5,3,10
14030 DATA ,3,2,4,9
14040 DATA ,5,1,5,8
14050 DATA ,3,1,6,7
14060 DATA " ",,,,
14070 DATA "S",,,,
14080 DATA "J",,,,
14090 DATA "I",,,,
14100 DATA ,5,4,11,14
14110 DATA ,5,5,12,13
14120 DATA "C",,,,
```

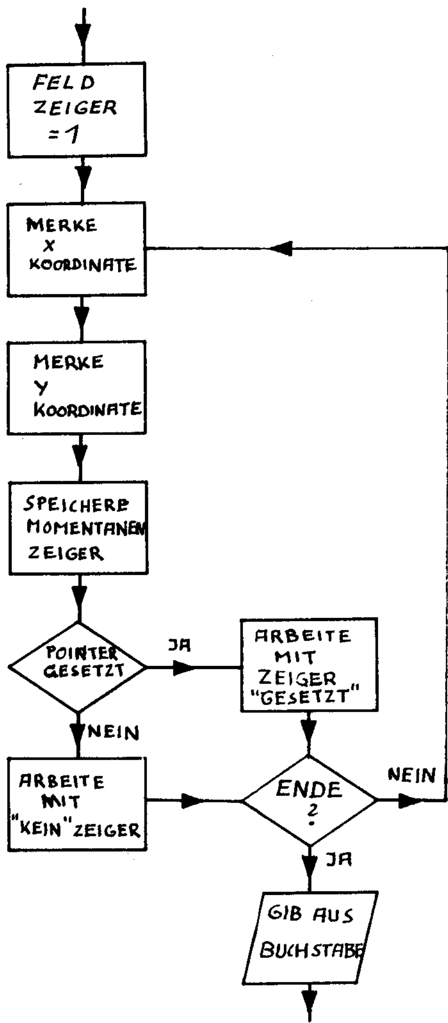
```
14130 DATA "G",,,,
14140 DATA ,5,7,18,15
14150 DATA ,2,4,17,16
14160 DATA "A",,,,
14170 DATA "Q",,,,
14180 DATA "O",,,,
14190 DATA ,5,1,20,29
14200 DATA ,5,4,21,29
14210 DATA ,5,3,27,22
14220 DATA ,5,7,23,26
14230 DATA ,5,5,24,25
14240 DATA "P",,,,
14250 DATA "B",,,,
14260 DATA "R",,,,
14270 DATA "L",,,,
14280 DATA "D",,,,
14290 DATA ,5,7,45,30
14300 DATA ,2,6,31,44
14310 DATA ,5,3,32,39
14320 DATA ,1,5,33,36
14330 DATA ,3,1,34,35
14340 DATA "X",,,,
14350 DATA "Z",,,,
14360 DATA ,4,2,38,37
14370 DATA "K",,,,
14380 DATA "E",,,,
14390 DATA ,2,4,40,43
14400 DATA ,4,2,42,41
14410 DATA "M",,,,
14420 DATA "N",,,,
14430 DATA "H",,,,
14440 DATA "W",,,,
14450 DATA ,3,1,46,51
14460 DATA ,1,5,47,50
14470 DATA ,2,4,48,49
14480 DATA "Y",,,,
14490 DATA "V",,,,
14500 DATA "U",,,,
14510 DATA ,1,5,52,53
14520 DATA "T",,,,
14530 DATA "F",,,,
```

Wenn Sie mehr Selbstvertrauen haben (oder Leerzeichen sparen wollen), können Sie alle DATA-Anweisungen in acht ziemlich unleserlichen Zeilen verdichten. Beachten Sie jedoch bitte die exakte Anzahl der Kommas, und seien Sie vorsichtig beim Ändern von Fehlern.

```
14010 DATA ,1,1,2,19,,1,5,3,10,,3,2,4,9,
,5,1,5,8,,3,1,6,7," ",,,,,"S",,,,
14080 DATA "J",,,,,"I",,,,,,5,4,11,14,,5
,5,12,13,"C",,,,,"G",,,,,,5,7,18,15
14150 DATA ,2,4,17,16,"A",,,,,"Q",,,,,"O
",,,,,,5,1,20,29,,5,4,21,28,,5,3,27,22
14220 DATA ,5,7,23,26,,5,5,24,25,"P",,,,
,"B",,,,,"R",,,,,"L",,,,,"D",,,,
14290 DATA ,5,7,45,30,,2,6,31,44,,5,3,32
,39,,1,5,33,36,,3,1,34,35,"X",,,,
14350 DATA "Z",,,,,,4,2,38,37,"K",,,,,"E
",,,,,,2,4,40,43,,4,2,42,41,"M",,,,
14420 DATA "N",,,,,"H",,,,,"W",,,,,,3,1,
46,51,,1,5,47,50,,2,4,48,49,"Y",,,,
14490 DATA "V",,,,,"U",,,,,,1,5,52,53,"T
",,,,,"F",,,,
```

Um nun ein beliebiges Zeichen gegen die vorhandenen Muster überprüfen zu können (siehe **Ablaufdiagramm 8.1**), muß der Tabellenzeiger AP zuerst auf 1 gesetzt werden, so daß die Überprüfung wirklich am Anfang beginnt. Die X- und Y-Koordinaten werden aus C1(AP) und C2(AP) gelesen, und der "letzte Position"-Zeiger LP wird auf den laufenden Wert des Tabellenzeigers AP gesetzt.
Die Punktfarbe PC dieser Koordinaten wird durch PEEK (TS + CO + X + (Y*40)) AND 15 festgestellt. Ist sie Null, dann ist der Punkt gesetzt, und wir müssen mit dem "Ja"-Zeiger Y(AP) weiterarbeiten. In jedem anderen Fall arbeiten wir mit dem "Nein"-Zeiger N(AP) weiter. In beiden Fällen muß jedoch überprüft werden, ob das angezeigte Element eine Null (die das Ende eines Entscheidungszweiges angeben würde) enthält, was ein erkanntes Zeichen bedeutet. In diesem Fall wird der entsprechende Buchstabe LE$(LP) ausgegeben. Die Bildschirmanzeige bleibt solange erhalten, bis durch Drücken einer Taste ein neuer Suchzyklus initiiert wird. Solange jedoch ein höherer Wert als Null gefunden wird, haben wir unseren Entscheidungsbaum noch nicht vollständig durchlaufen, das Programm springt zur Zeile 2010 zurück und arbeitet mit neuen Werten von C1(AP) weiter. Die überprüften Werte können Sie dadurch erkennen, daß diese nach dem Auffinden unterschiedliche Farben bekommen. "Ja"- und "Nein"-Verzweigungen werden nach

dem Testen, wenn sie nicht besetzt waren, hellgrün (3), und wenn sie gesetzt waren rot (3 + 1) dargestellt. Alle besetzten, aber nicht überprüften Punkte verbleiben schwarz.



Ablaufdiagramm 8.1 Zeichenerkennung

```
2000 AP=1
2010 X=C1(AP):Y=C2(AP):LP=AP
2020 PC=PEEK(TS+CO+X+(Y*40)) AND 15
2030 IF PC=0 THEN AP=Y(AP):GOTO2050
2040 AP=N(AP)
2050 IF AP=0 THEN 2070
2060 POKE TS+CO+X+(Y*40),3+(PC=0):GOTO 2
010
2070 PRINT LE$(LP);
2080 GET A$:IF A$="" THEN 2080
2090 GOSUB 13000:GOTO 20
```

Wenn Sie den wirklichen Entscheidungspfad wissen wollen, müssen Sie die folgenden Modifikationen in Ihr Programm einfügen, und die entsprechenden Informationen werden Ihnen ausgegeben. Durch Addition eines Offsets von 481 zu SS wird das Gitter an das untere Ende des Bildschirms verschoben, und mit BL$ legen wir uns eine Leerzeichenkette an, die wir später zum "Säubern" des Bildschirms benutzen.

```
10003 REM ZEILE 10005 40 MAL SPACEBALKEN
 DREUCKEN
10005 BL$="
          "

2005 PRINT"■";" AP":PRINT
2055 PRINT AP
2070 PRINT:PRINT " ";LE$(LP):PRINT
2075 PRINT"DRUECK EINE TASTE DANN GEHTS
WEITER"
2085 PRINT"■":FOR N=1 TO 10:PRINT BL$:NE
XT N
```

Diese Schnellsuchmethode, bei der nur kritische Entscheidungspunkte untersucht werden, hat einen erheblichen Nachteil. Im Falle eines im Entscheidungsbaum nicht enthaltenen Musters erhalten wir trotzdem eine Übereinstimmung, während bei der Methode, in der alle Punkte überprüft werden, keine Übereinstimmung erkannt wird. MIt den früheren Klarschriftlesern konnte man nur immer einen bestimmten Zeichensatz erkennen, während die neuesten Maschinen nicht nur verschiedene Zeichensätze zulassen, sondern auch durch ein eingebautes Expertensystem die

dafür notwendigen Erkennungsregeln selbst lernen. Sie können sie dies lehren, indem Sie ihnen einige Textseiten zeigen und dann die gleichen Zeichen über die Tastatur eingeben. Es wird aber bestimmt noch eine lange Zeit dauern, bis jemand eine Maschine gebaut hat, die unsere Handschrift lesen kann.

KAPITEL 9

EIN INTELLIGENTER LEHRER

Ein weiterer Anwendungsbereich, in dem künstliche Intelligenz sehr nützlich sein kann, sind die Lehrprogramme. Es wäre schon sehr gut, ein Programm zu haben, das das Wissen eines Schülers wahllos abfragt; aber das ist nicht die Arbeitsweise eines wirklichen Lehrers. Neben dem Stellen von Fragen beobachtet er den Fortschritt des Schülers, steigert die Schwierigkeit der Fragen mit wachsender Erfahrung und prüft dann speziell in den Bereichen, in denen er Schwierigkeiten hat. Wenn z. B. ein Kind in den Bereichen Addieren, Subtrahieren, Multiplizieren und Dividieren überprüft wird und nur die Divisionen falsch macht, dann müssen diesem Kind in der Zukunft zur Übung mehr Divisionsaufgaben gegeben werden. Lassen Sie uns nun sehen, wie wir diese "menschlichen" Qualitäten in ein Lehrprogramm einbauen können.

FRAGEN UND ANTWORTEN

Zuerst benötigen wir Zufallszahlen ("Random"-Zahlen), die wir in der Addition benutzen. Die Funktion INT(RND(1)*10) gibt uns Zahlen zwischen 0 und 9.

```
20 A%=INT(RND(1)*10)
30 B%=INT(RND(1)*10)
```

Der Computer addiert diese und geht dann zu einem "Eingabe- und Prüf"-Programm bei Zeile 1000.

```
40 C%=A%+B%:GOSUB 1000
```

Zuerst muß die Frage ausgegeben werden, danach wird Ihre Antwort IP$ abgefragt.

```
1000 PRINT A%;"+";B%;"=";
1010 INPUT IP%
```

Ihre Antwort muß dann überprüft werden. Wenn die Programmantwort C% mit Ihrer Antwort übereinstimmt, wird RICHTIG ausgegeben, und das Programm kehrt zur Zeile 40 zurück. Anderenfalls wird FALSCH, DIE RICHTIGE ANTWORT IST gefolgt von der richtigen Antwort ausgegeben.

```
1020 IF C%=IP% THEN IN%=-1:PRINT "RICHTI
G":RETURN
1030 IN%=1:PRINT "FALSCH, DIE RICHTIGE A
NTWORT HEISST ";C%
1040 RETURN
```

Mit den anderen drei Bereichen (Subtraktion, Multiplikation und Division) kann man in der gleichen Art und Weise verfahren. Dazu ersetzen wir das "Plus"-Zeichen 1000 durch SG$, das dann für die entsprechende Aufgabe immer mit dem passenden Rechenoperator versehen wird. Da INT(RND(1)*10) bei allen Berechnungen benutzt wird, definieren wir es uns als Funktion RD.

```
15 DEF FNRD(X)=RND(1)*10
20 A%=FNRD(AD%)
30 B%=FNRD(AD%)
40 SG$="+":C%=A%+B%:GOSUB 1000
50 A%=FNRD(SU%)
60 B%=FNRD(SU%)
70 SG$="-":C%=A%-B%:GOSUB 1000
80 A%=FNRD(MU%)
90 B%=FNRD(MU%)
100 SG$="*":C%=A%*B%:GOSUB 1000
110 B%=FNRD(DI%)+1
120 A%=INT(FNRD(DI%))*B%
130 SG$="/":C%=A%/B%:GOSUB 1000
1000 PRINTA%;SG$;B%;"=";
```

Schließlich springen wir zur Zeile 20 zurück, um neue Aufgaben zu stellen.

```
140 GOTO20
```

DIVISION DURCH NULL!

So wie das Programm jetzt arbeitet, wird es bei einer Division durch 0 "abstürzen". Dieses Problem können wir einfach dadurch lösen, daß wir zu dem Divisor B% immer eine 1 addieren.

```
120 B%=FNRD(DI%)+1
```

WIR VERMEIDEN DEZIMALZAHLEN

Obwohl wir in den Aufgaben nur ganze Zahlen benutzen, kann das Ergebnis einer Division natürlich eine Dezimalzahl sein. Um dieser Problematik von vornherein aus dem Wege zu gehen, müssen wir dafür sorgen, daß A% ein Mehrfaches von B% ist. Aus diesem Grunde berechnen wir erst B% und berechnen A% aus einer Multiplikation von B% mit einer Zufallszahl zwischen 0 und 10.

```
110 B%=FNRD(DI%)+1
120 A%=INT(FNRD(DI%))*B%
```

WIR ZÄHLEN DIE RICHTIGEN ANTWORTEN

Nachdem der Test selbst einwandfrei arbeitet, sollten wir uns um das Zählen der richtigen Antworten kümmern. Am einfachsten zählen wir eine "Versuch"-Variable TR% bei jeder Benutzung des Unterprogramms bei der Zeile 1000 hoch und tun das gleiche mit einer "Treffer"-Variablen SC% bei jeder richtigen Antwort.

```
1010 INPUT IP%:TR%=TR%+1
1020 IF C%=IP% THEN:PRINT "RICHTIG":SC%=
SC%+1:GOTO 1040
1040 PRINT "DU ERREICHTEST ";SC%;"/";TR%
:RETURN
```

Wenn Sie den prozentualen Anteil der richtigen Antworten von der Gesamtanzahl der Fragen bevorzugen, sollten Sie die Zeile 1040 wir folgt abändern:

```
1040 PRINT "DU ERREICHTEST ";INT((SC%/TR
%)*100);"% RICHTIGE ANTWORTEN":RETURN
```

WIE VIELE FRAGEN SOLLEN WIR STELLEN?

Das jetzige Programm fragt nacheinander jeweils eine Frage eines Typs und fängt dann wieder von vorne an. Durch die Definition der erlaubten Anzahl von Fragen als Variable NQ% können wir die Anzahl der zu stellenden Fragen begrenzen.

```
10 NQ%=32
```

Bei jeder Frage wird NQ% um 1 vermindert, und der Test wird beendet, wenn NQ%=0 erreicht ist.

```
150 IF NQ%>0 THEN 20
160 END
1010 INPUT IP%:TR%=TR%+1:NQ%=NQ%-1
```

WIR VERSCHIEBEN DIE SCHWERPUNKTE

Wenn wir jedoch bei den Fragen Schwierigkeitsbereiche berücksichtigen wollen, müssen wir uns den Leistungsstand für jeden Bereich merken. Wir brauchen deshalb für jeden Fragentyp getrennte Variablen (AD% für Addition, SU% für Subtraktion, MU% für Multiplikation und DI% für Division). Diese Variablen werden als der achte Teil der Anzahl der Fragen NQ% definiert.

```
10 NQ%=32:AD%=NQ%/8:SU%=AD%:MU%=AD%:DI%=
AD%:WT%=NQ-(2*AD%)
```

Wenn nun die richtige Antwort C% mit Ihrer Antwort IP% übereinstimmt, wird IN% auf −1 gesetzt, RICHTIG ausgegeben, und das Unterprogramm kehrt zurück. Im anderen Fall wird I% auf 1 gesetzt und FALSCH, gefolgt von der richtigen Antwort ausgegeben.

```
1020 IF C%=IP% THEN IN%=-1:PRINT"RICHTIG
":RETURN
1030 IN%=1:PRINT"FALSCH. RICHTIGE ANTWOR
T WAR ";C%
1040 RETURN
```

Danach wird IN% zu der entsprechenden Fragevariablen (AD%, SU%, MU% oder DI%) addiert. Bei einer falschen Antwort wird dadurch die Frage-Variable erhöht, bei einer richtigen Antwort vermindert.

```
40 SG$="+":C%=A%+B%:GOSUB 1000:AD%=AD%+I
N%
70 SG$="-":C%=A%-B%:GOSUB 1000:SU%=SU%+I
N%
100 SG$="*":C%=A%*B%:GOSUB 1000:MU%=MU%+
IN%
```

```
130 SG$="/":C%=A%/B%:GOSUB 1000:DI%=DI%+
IN%
```

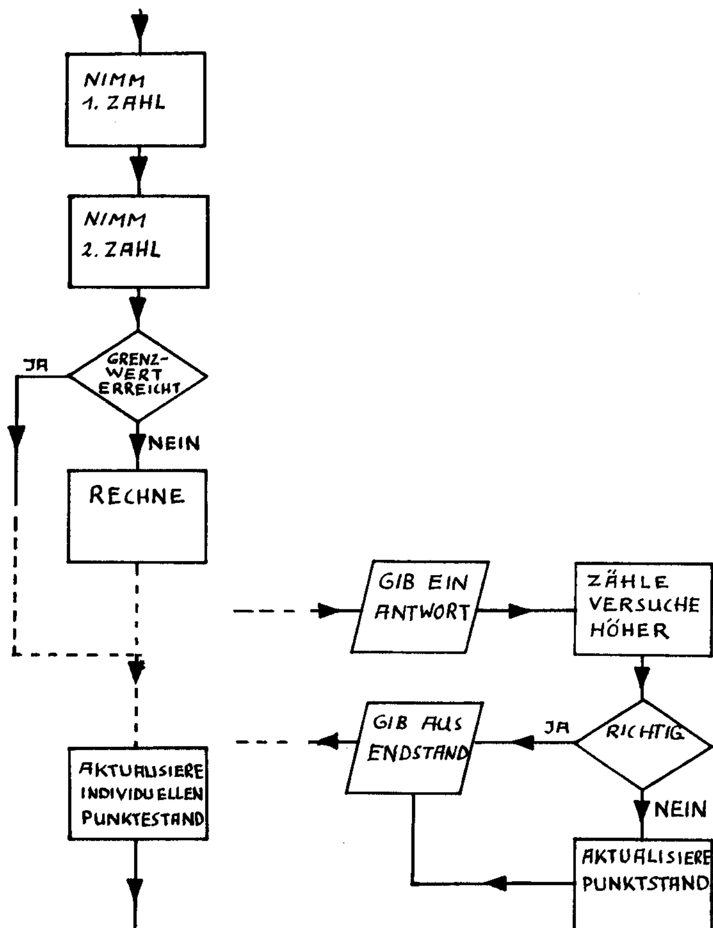
Nun erweitern wir unser Programm um die Überprüfung, ob alle Fragen eines speziellen Typs fehlerhaft beantwortet wurden (z. B. AD%>0, siehe **Ablaufdiagramm 9.1**). Wenn alle Fragen eines Typs korrekt beantwortet worden sind, werden keine Fragen dieser Art mehr gestellt, da die entsprechende Programmzeile übersprungen wird. Wenn jedoch die vorgegebene Anzahl der Fragen jeden Typs korrekt beantwortet wurden (AD%=0, SU%=0, MU%=0 und DI%=0), wird das Programm beendet.

```
40 IF AD%>0 THEN SG$="+":C%=A%+B%:GOSUB
1000:AD%=AD%+IN%
70 IF SU%>0 THEN SG$="-":C%=A%-B%:GOSUB
1000:SU%=SU%+IN%
100 IF MU%>0 THEN SG$="*":C%=A%*B%:GOSUB
 1000:MU%=MU%+IN%
130 IF DI%>0 THEN SG$="/":C%=A%/B%:GOSUB
 1000:DI%=DI%+IN%
140 IF AD%=0 AND SU%=0 AND MU%=0 AND DI%
=0 THEN 160
```

Wenn Sie in einem Bereich vier Fragen ohne Fehler beantwortet haben, werden hier keine weiteren Fragen gestellt. Wenn Sie jedoch einen Fehler machen, werden die entsprechenden Variablen (AD% usw.) wieder erhöht, und Sie müssen mehr als vier korrekte Antworten geben, bevor Sie wieder den Wert 0 erreichen.

WIR VARIIEREN DIE SCHWIERIGKEIT

Es wäre bestimmt sehr interessant, die Schwierigkeit der Fragen entsprechend der Qualität der Antworten zu variieren. Bis jetzt schwanken die Werte für A% und B% immer zwischen 0 und 9, da sie mit der Funktion RND(1)*10 erstellt wurden. In der Zukunft wollen wir im Fall einer richtigen Antwort größere Zahlen benutzen, im Fall einer falschen Antwort jedoch kleinere. Dabei müssen wir jedoch verhindern, daß wir im Fall einer katastrophalen Leistung in den negativen Bereich abrutschen.
Im Grenzfall könnten Sie alle Fragen der ersten drei Gruppen richtig beantwortet haben, aber keine der letzten Gruppe. In diesem Fall wären nur vier Fragen für jede der ersten drei Gruppen gestellt worden, wodurch 32−(3*+4)=20 Fragen für die letzte Gruppe verbleiben würden. Außerdem müssen wir berücksichtigen, daß X



Ablaufdiagramm 9.1 Intelligenter Lehrer

(z. B. AD%) mit dem Wert 4 beginnt, so daß der höchste·erreichbare Wert von X 20+4=24 ist.
Wir definieren uns deshalb eine Wichtungsvariable WT%, die durch die Subtraktion der dreifachen Anzahl der in einer Gruppe zu stellenden Fragen (3*AD%) von der Gesamtzahl der zu fragenden Antworten NQ%, erhöht um die Anzahl der Fragen einer Gruppe AD% errechnet, wird.

WT%=NQ%-(3*AD%)+AD%

Vereinfacht sieht das so aus:

WT%=NQ%-(2*AD%)

```
10 NQ%=32:AD%=NQ%/8:SU%=AD%:MU%=AD%:DI%=
AD%
```

Wir ersetzen nun den Wert 10 in unserer Random-Funktion durch die Differenz zwischen WT% und X.

```
15 DEF FNRD(X)=RND(1)*(WT%-X)
```

Wir beginnen mit WT%=24 und X=4, so daß Zahlen zwischen 0 und 19 gewählt werden können. Bei einer richtigen Antwort wird X auf 3 reduziert, so daß Zahlen zwischen 0 und 20 gewählt werden können. Nach vier korrekten Antworten ist X=0, und diese Zeile wird übersprungen. Die letzten möglichen Werte liegen demnach zwischen 0 und 22. Wenn jedoch bereits die erste Antwort falsch war, wird X um 1 vermindert (0 bis 18). Im Grenzfall wird X ingesamt zwanzigmal auf 24 erhöht und (WT%-X) wird für A% und B% auf 0 fallen. Dieses spezielle Problem müßten sie noch bei Ihrem Programm berücksichtigen.

KAPITEL 10

DAS GESAMTPROGRAMM

In den vorherigen Kapiteln haben wir, beginnend mit den einfachsten Prinzipien, die verschiedenen Aspekte der künstlichen Intelligenz behandelt. In diesem abschließenden Kapitel haben wir viele dieser Einzelideen in einem Gesamtprogramm vereinigt.
Das erste "intelligente" Programm überhaupt war das berühmte "ELIZA", ein pseudo-psychiatrisches Programm zur Erstellung einer bestimmten psychiatrischen Therapie. Wir haben jedoch der Versuchung, diesem Weg zu folgen, widerstanden und haben uns indessen entschlossen, ein Programm für einen Computerverkäufer zu erstellen. Unter Benutzung einiger Ideen aus den Bereichen der Sprachverarbeitung und der Expertensysteme ist dieses Programm in der Lage, ein Ergebnis zu erstellen, das auf dem Verständnis Ihrer Anforderungen beruht und Empfehlungen zu geben, die Ihre Anforderungen und wirtschaftlichen Fakten berücksichtigen.
Wir haben bis jetzt schon eine Menge Wörter und Werte berücksichtigt, die das Programm recht interessant machen, aber Sie können es durch Hinzufügen weiterer DATA-Anweisungen noch auf Ihre Anforderungen maßschneidern. (Wir können jedoch keine Verantwortung für die im Programm enthaltenen Werte übernehmen, da sie nur für diese Demonstration erstellt wurden.) Das Programm ist ziemlich komplex, aber es basiert auf den bis jetzt in diesem Buch beschriebenen Methoden. Die Funktion der verschiedenen Variablen und Tabellen ist in der **Tabelle 10.1** erklärt.

WIR UNTERHALTEN UNS

Die Aufgabe des Programms ist es, Sie über Ihre Ansichten in bezug auf eine Anzahl von möglichen Eigenschaften zu befragen. Die genaue Frage wird willkürlich aus einer Gruppe von Redewendungen zusammengestellt. Beachten Sie, daß das Schlüsselwort bzw. die Redewendung unter Anwendung der richtigen Konjugation (wenn notwendig) in den Satz eingefügt wird.
Ihre Eingabe wird im Detail auf Schlüsselwörter untersucht und eine Regeltabelle wird entsprechend Ihrer Anforderungen verändert. Wenn Sie beobachten wollen, wie die Regeltabelle verändert wird, dann löschen Sie die Zeile 5490. Viele der Schlüsselwörter sind verkürzt, so daß mit einer Überprüfung viele gleichartige Wörter gefunden werden können, und mit einem weiteren Test wird überprüft, ob die gefundene Zeichenkette am Anfang eines Wortes steht.

EINFACHE VARIABLEN

IS	INSTR-Beginn
I1$	zu überprüfende Zeichenkette
I2$	zum Überprüfen benutzte Zeichenkette
IP	INSTR-Zeiger
QP	Anzahl der Fragesätze
Q	Anzahl der Fragen
R	Anzahl der Regeln
BB	Kontostand
PH	Anzahl der Redewendungen
PH$	Anzahl der Wörter einer Redewendung
M	Übereinstimmungsmarkierung
OF	Objektkennzeichen
OM	Objektübereinstimmung
LD	Zustimmung/Ablehnung
FS	Satzrest-Zeiger
NP	Negativzeiger
S1	UND Übereinstimmungszeiger
S2	ABER Übereinstimmungszeiger
RU	Regeländerungsmarkierung
OB	Anzahl der Objekte
AJ	Anzahl der Adjektive
AV	Anzahl der Adverbien
LI	Anzahl der Zustimmungen
DL	Anzahl der Ablehnungen
NJ	Anzahl der negativen Adjektive
NV	Anzahl der negativen Adverbien
HM	Anzahl von "billig/teuer"
CO	Anzahl der Computer
FE	Anzahl der Eigenschaften
CT	Anzahl der Kostenvergleiche
CS	Anzahl der Kostenempfehlungen
EX	Anzahl der Entschuldigungen
HI	Anzahl der Hochpreisempfehlungen
LO	Anzahl der Niedrigpreisempfehlungen
TC	Gesamtkosten
TP	Gesamtgewinn

Tabelle 10.1 Die wichtigsten Variablen in "Verkäufer"

TABELLEN (FELDER)

OB$(OB)	Objekte
AJ$(AJ)	Adjektive
NJ$(NJ)	negative Adressen
AV$(AV)	Adverbien
NV$(NV)	negative Adverbien
LI$(LI)	Zustimmungen
DL$(DL)	Ablehnungen
Q$(Q)	Frageobjekte
QP$(QP)	Fragesätze
CR(Q)	Kosten
PR(Q)	Gewinn
IC(Q)	Gesamtkosten
IP(Q)	Gesamtgewinn
HM$(HM)	billig/teuer
R(R)	Regeln
CO$(FE)	Computernamen
FE(CO,FE)	Eigenschaftsnamen
C(CT)	Kostenvergleiche
CS$(CS)	Kostenempfehlungen
EX$(EX)	Entschuldigungen
HI$(HI)	"teuer" Informationen
LO$(LO)	"billig" Informationen

Mit der einfachsten Antwort "JA" oder "NEIN" wird jeweils eine 1 zu der entsprechenden Regel für diese Eigenschaften addiert oder davon subtrahiert. Wenn Sie den Namen einer Eigenschaft erwähnen (z. B. "GRAPHIK"), wird eine weitere Regel addiert. Außerdem erhöht ein "positives Adjektiv oder Adverb die Regel, während ein "negatives" Adjektiv oder Adverb die Regel vermindert. Durch die Aufteilung der Wörter in verschiedene Klassen können Sie zur gleichen Zeit mehr als eine Änderung an einer Regel durchführen.
So ergibt:

JA	plus 1
JA BASIC	plus 2
JA BASIC NOTWENDIG	plus 3

JA GUTES BASIC NOTWENDIG plus 4

Dagegen gibt:

NEIN minus 1

KEIN SPEICHER minus 2

Darüber hinaus werden Verben in "Zustimmungen" und "Ablehnungen" unterteilt, wobei durch "Ablehnungen" der Sinn des Rests der Wörter umgekehrt wird.
Deshalb ergibt:

ICH VERABSCHEUE MACRODRIVES minus 1

"NEIN" und "NICHT" werden erkannt, und die meisten doppelten Negierungen werden korrekt interpretiert.
Somit ergibt:

ICH LEHNE TON AB minus 2

ICH LEHNE TON NICHT AB plus 1

Eine Information am Anfang des Satzes, die von einem Komma gefolgt wird, wird normalerweise abgeschnitten und nicht mehr berücksichtigt. D. h.:

NEIN ICH MOECHTE KEINEN GUTEN KLANG minus 3

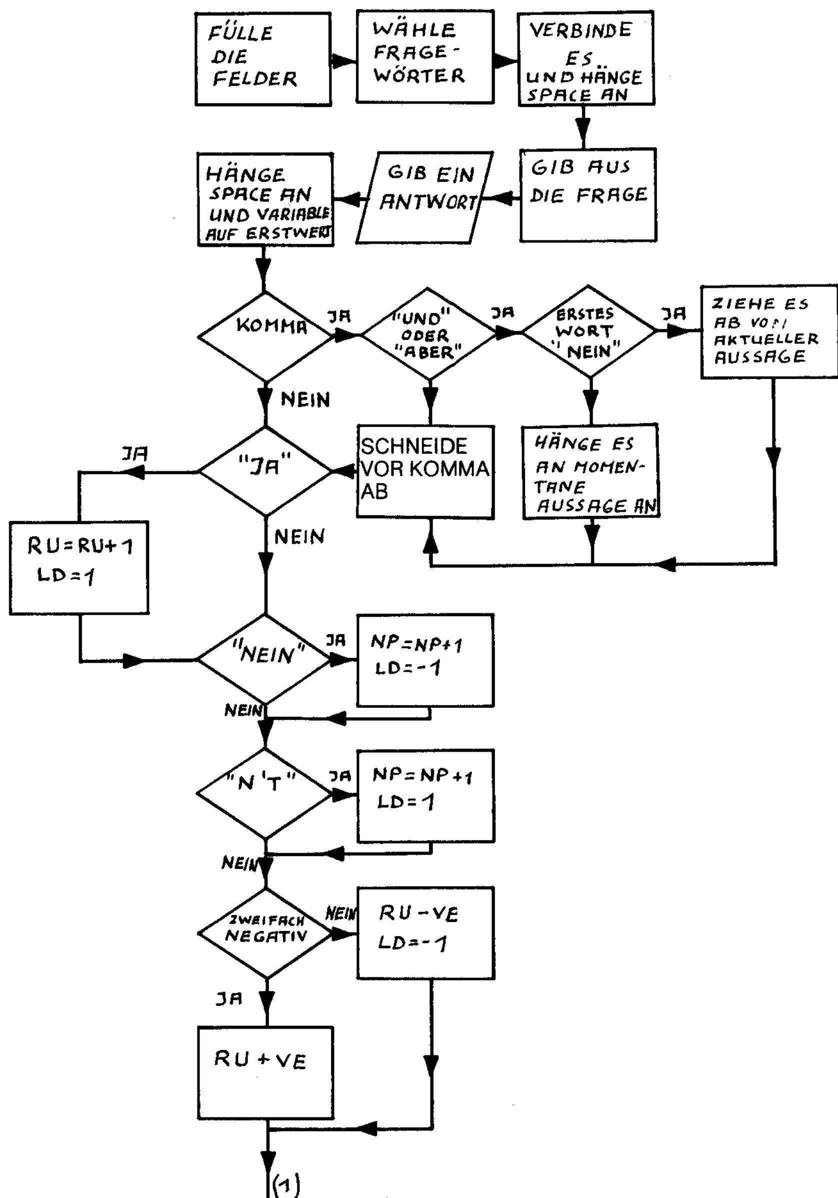
Eine Ausnahme wird gemacht, wenn "UND" oder "ABER" enthalten sind, und wenn beide Satzteile unterschiedliche Meinungen ausdrücken. Das bedeutet, wenn die Frage:

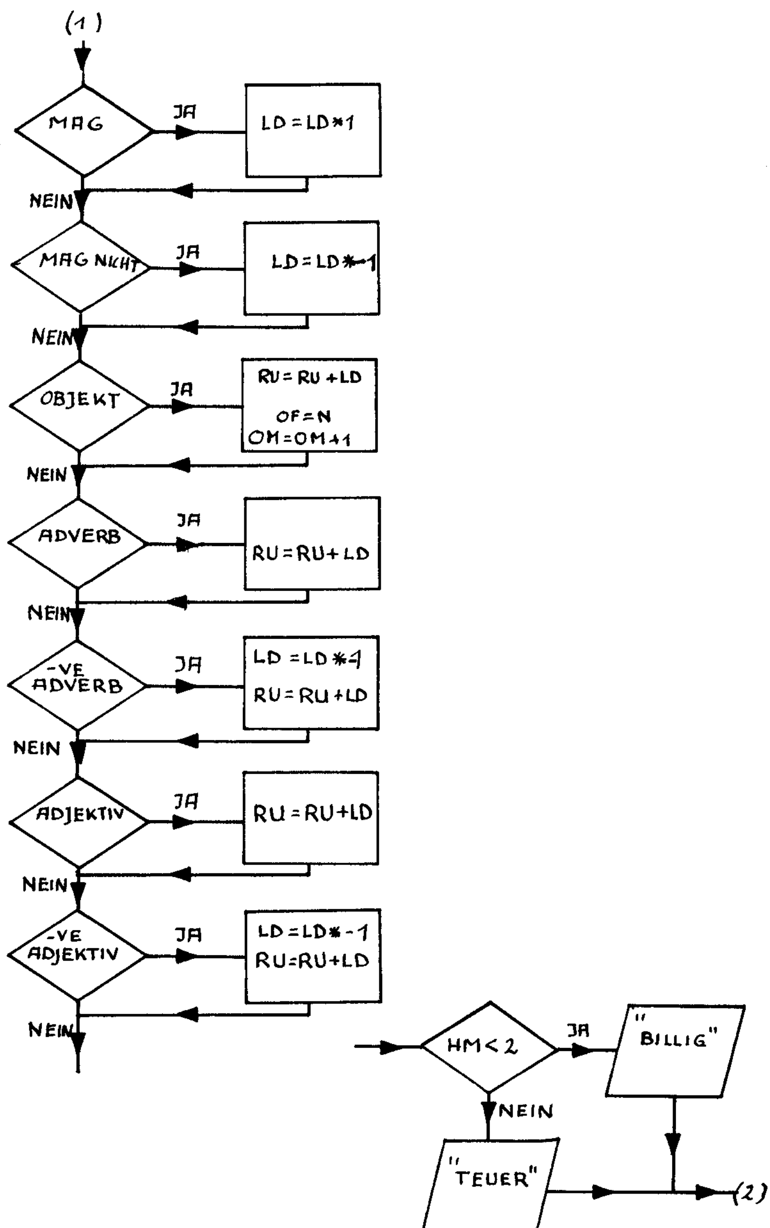
MOECHTEN SIE GRAFIK?

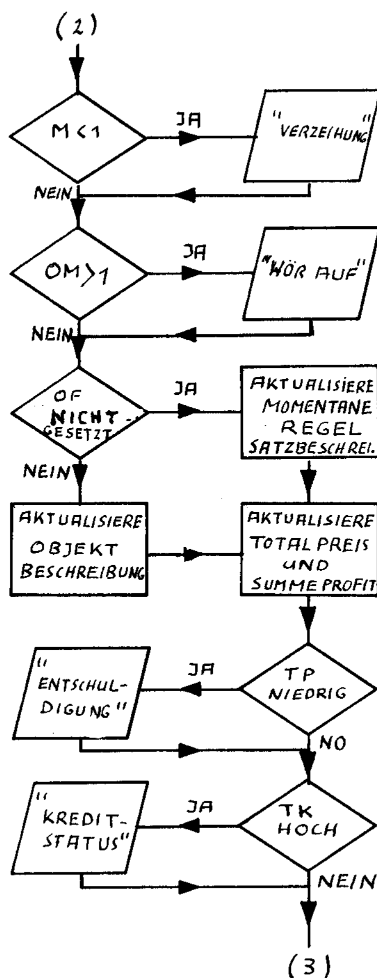
beantwortet wird mit:

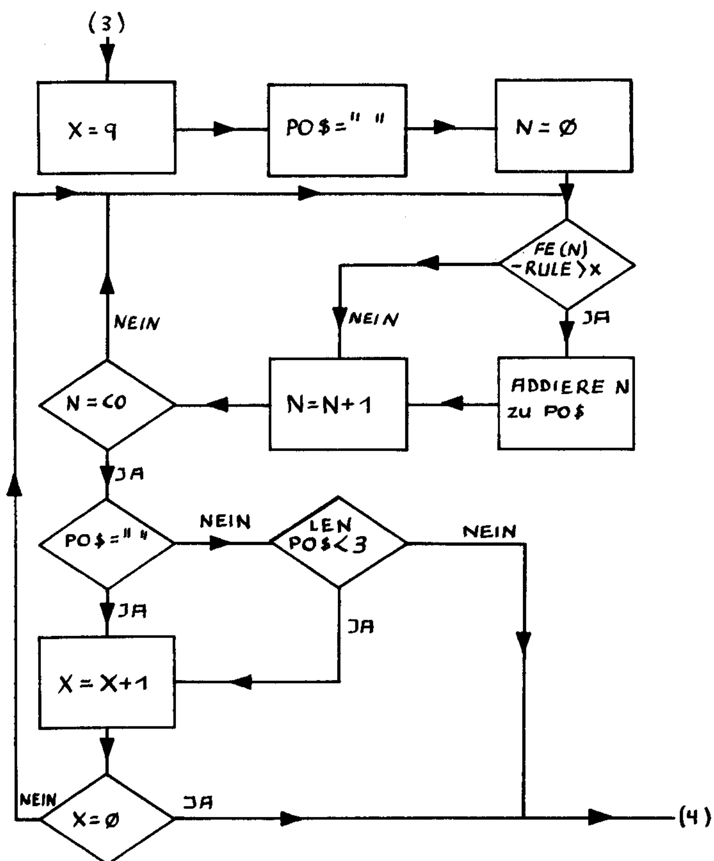
NEIN, ABER ICH MOECHTE GUTEN KLANG

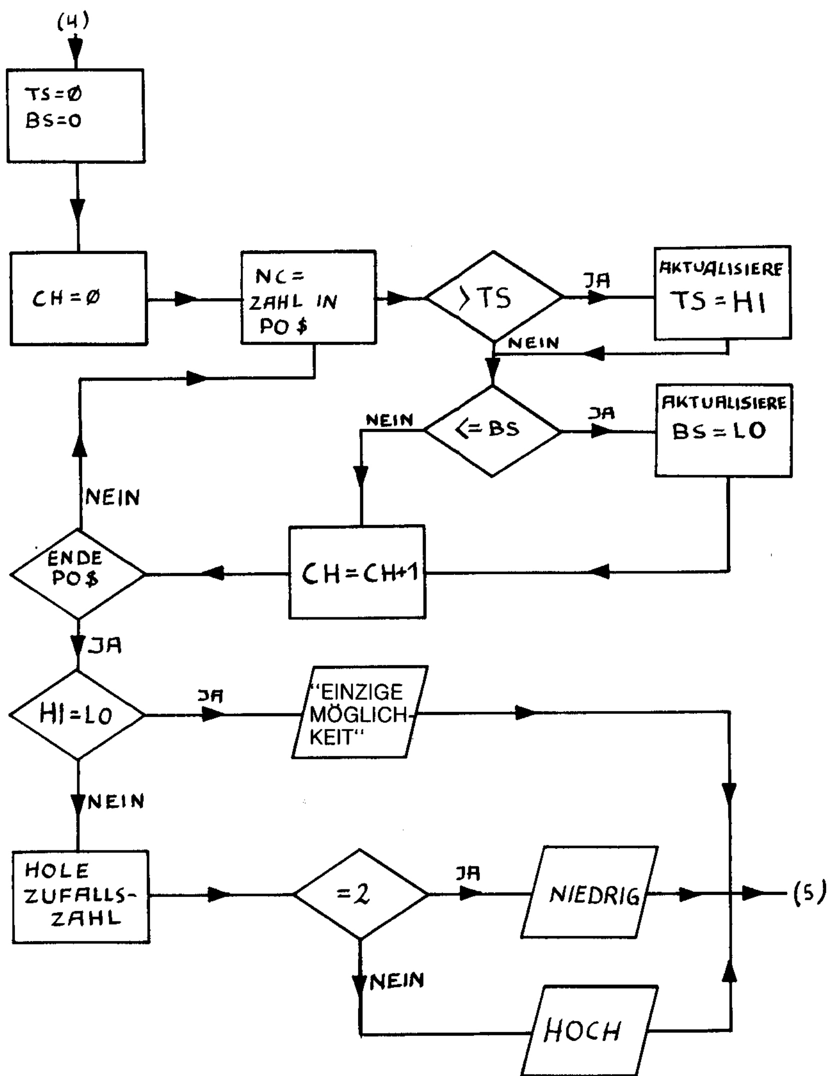
dann wird 1 von der Grafikregel subtrahiert, und zwei zur Klangregel addiert.
Wenn das Programm kein einziges Schlüsselwort in der Eingabe findet, fragt es höflich nach einem neuen Versuch:


FÜLLE DIE FELDER
WÄHLE FRAGE-WÖRTER
VERBINDE ES UND HÄNGE SPACE AN
GIB AUS DIE FRAGE
GIB EIN ANTWORT
HÄNGE SPACE AN UND VARIABLE AUF ERSTWERT
KOMMA
JA
"UND" ODER "ABER"
JA
ERSTES WORT "NEIN"
JA
ZIEHE ES AB VOM AKTUELLER AUSSAGE
NEIN
SCHNEIDE VOR KOMMA AB
HÄNGE ES AN MOMENTANE AUSSAGE AN
JA
"JA"
RU=RU+1
LD=1
NEIN
"NEIN"
JA
NP=NP+1
LD=-1
NEIN
"N'T"
JA
NP=NP+1
LD=1
NEIN
ZWEIFACH NEGATIV
NEIN
RU-VE
LD=-1
JA
RU+VE
(1)


(1)
MAG
JA
LD=LD*1
NEIN
MAG NICHT
JA
LD=LD*-1
NEIN
OBJEKT
JA
RU=RU+LD
OF=N
OM=OM+1
NEIN
ADVERB
JA
RU=RU+LD
NEIN
-VE ADVERB
JA
LD=LD*-1
RU=RU+LD
NEIN
ADJEKTIV
JA
RU=RU+LD
NEIN
-VE ADJEKTIV
JA
LD=LD*-1
RU=RU+LD
NEIN
HM<2
JA
"BILLIG"
NEIN
"TEUER"
(2)


(2)
M<1
JA
"VERZEIHUNG"
NEIN
OM>1
JA
"HÖR AUF"
NEIN
OF NICHT-GESETZT
JA
AKTUALISIERE MOMENTANE REGEL SATZBESCHREI.
NEIN
AKTUALISIERE OBJEKT BESCHREIBUNG
AKTUALISIERE TOTAL PREIS UND SUMME PROFIT
TP NIEDRIG
JA
"ENTSCHUL-DIGUNG"
NO
TK HOCH
JA
"KREDIT-STATUS"
NEIN
(3)


(3)
X = 9
PO$ = " "
N = Ø
FE(N) -RULE > X
NEIN
JA
ADDIERE N zu PO$
N=N+1
N = C0
NEIN
JA
PO$ = " "
NEIN
LEN PO$ < 3
NEIN
JA
JA
X = X+1
NEIN
X = Ø
JA
(4)



(4)
TS=Ø
BS=0
CH=Ø
NC=
ZAHL IN
PO $
>TS
JA
AKTUALISIERE
TS=HI
NEIN
<=BS
JA
NEIN
AKTUALISIERE
BS=LO
NEIN
ENDE
PO $
CH=CH+1
JA
HI=LO
JA
"EINZIGE
MÖGLICH-
KEIT"
NEIN
HOLE
ZUFALLS-
ZAHL
=2
JA
NIEDRIG
(5)
NEIN
HOCH

VERZEIHUNG, ENTSCHULDIGEN SIE MICH ABER . . .

Das Programm kann nicht gleichzeitig zwei Funktionen behandeln. Wenn Sie z. B. zur gleichen Zeit nach "KLANG und GRAFIK" fragen, werden Sie zur Wiederholung der Frage aufgefordert.

HALT DIE LUFT AN, EINS NACH DEM ANDEREN

Sie haben jedoch die Möglichkeit, Kommentare in bezug auf einzelne Eigenschaften zu machen, obwohl danach im Moment nicht gefragt wird. Durch diese Eingaben werden die Regeln trotzdem entsprechend geändert (wie bei dem "ABER"-Beispiel im obigen Satz).

ENTSCHEIDUNGEN

Neben der Regeltabelle gibt es zwei weitere Tabellen, die mit dieser verknüpft sind. Die "Kostentabelle" enthält Informationen über die Kosten einer speziellen Aktion, und die "Gewinntabelle" gibt dem Verkäufer Information darüber, ob es sich lohnt, einen bestimmten Aufwand für den Verkauf zu tätigen. Die Werte dieser beiden Tabellen werden durch die Multiplikation des Inhalts des entsprechenden Regeltabellenelements mit den als DATA-Anweisungen in Zeile 10100 eingegebenen Faktoren berechnet. Das Format dieser DATA-Anweisungen ist:

(Name der Eigenschaft, Kosten, Gewinn).

Nach jeder Eingabe berücksichtigt der Verkäufer die Konsequenzen Ihrer Anforderungen. Als erstes überprüft er, ob die Gesamtkosten bei Ihren Anforderungen Ihr Guthaben übersteigt. Wenn dies der Fall ist, werden einige sarkastische Kommentare in bezug auf Ihre Kreditwürdigkeit ausgegeben:

DIESE AUSFUEHRUNG SCHEINT IHREN KREDITRAHMEN ZU SPRENGEN

Er überprüft auch die Höhe seines möglichen Gewinns bei diesem Geschäft; wenn dieser zu gering ist, wird er das Interesse verlieren und sich mit folgenden Argumenten herausreden:

MOMENT BITTE DAS TELEFON KLINGELT, ICH HABE EINEN WICHTIGEN TERMIN

oder

WIR SCHLIESSEN IN FUENF MINUTEN

Daneben ist er jedoch sehr hilfreich. Er stellt Vergleichslisten in bezug auf Ihre Anforderungen auf, um Ihnen mitzuteilen, welcher der verfügbaren Computer Ihren Bedürfnissen gerecht wird. Dabei vergleicht er den vom Computer-Hersteller angenommenen Wert einer Eigenschaft mit Ihren Forderungen. Das Format dieser Vergleichsinformationen ist:

(Name, Wert der Eigenschaft 1, Wert der Eigenschaft 2, Wert der Eigenschaft 3 usw.)

Wenn möglich, wird die am höchsten bewertete Maschine immer zuerst ausgewertet. Auf jeden Fall werden jedoch drei Maschinen (möglicherweise mit niedrigeren Bewertungen) ausgewählt, und die endgültige Entscheidung wird daraus getroffen. Der billigste oder der teuerste Computer wird wahllos herausgesucht und z. B. mit folgenden Bemerkungen angeboten:

WOLLEN SIE ABER DEN ROLLS-ROYCE DER COMPUTER DANN IST ES DER . . .

und

WENN SIE NICHT SOVIEL AUSGEBEN MOECHTEN – WIE IST ES DANN MIT . . .

Wenn jedoch nur eine Maschine in den vorgegebenen finanziellen Rahmen paßt, wird das Programm mit folgender Empfehlung aufwarten:

IHRE EINZIGE MOEGLICHKEIT IST . . .

Verkäufer

```
10 GOSUB 9300
20 GOTO 200
100 FOR IS=FT TO LEN(I1$)
110 IF MID$(I1$,IS,LEN(I2$))=I2$ THEN IP
=IS:RETURN
120 NEXT IS
130 IP=0:RETURN
200 PH=INT(RND(1)*(QP)+1):PH$=QP$(PH)
210 I1$=PH$:I2$="/":FT=1:GOSUB 100:SP=IP
```

```
220 IF SP=0 THEN 400
230 IFLEFT$(Q$(Q),1)="@"THENPH$=LEFT$(PH
$,SP-1)+"  SIND"+RIGHT$(PH$,LEN(PH$)-SP)

310 IFLEFT$(Q$(Q),1)="&"THEN PH$=LEFT$(P
H$,SP-1)+"  IST"+RIGHT$(PH$,LEN(PH$)-SP)

400 I1$=PH$:I2$="*":FT=1:GOSUB 100
410 IF SP=0 THEN 440
420 PH$=LEFT$(PH$,SP-1)+" "+RIGHT$(Q$(Q)
,LEN(Q$(Q))-1)+RIGHT$(PH$,LEN(PH$)-SP)
430 GOTO 500
440 PH$=PH$+" "+RIGHT$(Q$(Q),LEN(Q$(Q))-
1)
450 PRINT:PRINT
500 PRINT PH$;" ?"
600 PRINT
700 IN$=" "
710 GET I$:PRINT "<█";
720 IF I$="" THEN 710
730 IF I$=CHR$(13) THEN 800
740 IN$=IN$+I$
750 PRINT I$;
760 GOTO 710
800 LD=1:OF=-1:FS=1:NP=0:RU=0:M=0:OM=0:S
1=0:S2=0
900 I1$=IN$:I2$=",":FT=1:GOSUB 100:CM=IP

910 IF CM=0 THEN 1600
1000 I1$=IN$:I2$="UND":FT=1:GOSUB 100:S1
=IP
1010 I1$=IN$:I2$="ABER":FT=1:GOSUB 100:S
2=IP
1200 IF S1+S2=0 THEN 1500
1300 IF LEFT$(IN$,5)<>" NEIN" THEN 1400
1310 R(Q)=R(Q)-1:IC(Q)=IC(Q)-CR(Q):IP(Q)
=IP(Q)-PR(Q):GOTO 1500
1400 R(Q)=R(Q)+1:IC(Q)=IC(Q)+CR(Q):IP(Q)
=IP(Q)+PR(Q)
1500 IN$=RIGHT$(IN$,LEN(IN$)-CM)
```

```
1600 I1$=IN$:I2$="JA":FT=FS:GOSUB 100:SP
=IP
1700 IF SP>0 THEN RU=RU+1:LD=1:M=1:FS=SP
+1:GOTO 1600
1800 I1$=IN$:I2$="NEIN":FT=FS:GOSUB 100:
SP=IP
1900 IF SP>0 THEN LD=-1:M=1:FS=SP+1:NP=N
P+1:GOTO 1800
2000 I1$=IN$:I2$="NICHT":FT=FS:GOSUB 100
:SP=IP
2100 IF SP>0 THEN LD=-1:M=1:FS=SP+1:NP=N
P+1:GOTO 2000
2200 IF NP=0 THEN 2300
2210 IF INT(NP/2)=NP/2 THEN RU=RU+1:LD=1

:GOTO 2300
2250 RU=RU-1:LD=-1
2300 FOR N=0 TO LI
2400 I1$=IN$:I2$=LI$(N):FT=1:GOSUB 100:S
P=IP
2410 IF SP=0 THEN 2500
2420 IF MID$(IN$,SP-1,1)=" " THEN LD=LD+
1:M=1
2500 NEXT N
2600 FOR N=0 TO DL
2700 I1$=IN$:I2$=DL$(N):FT=1:GOSUB 100:S
P=IP
2710 IF SP>0 THEN IF MID$(IN$,SP-1,1)="
" THEN LD=LD*-1:M=1
2800 NEXT N
2900 FOR N=0 TO OB
3000 I1$=IN$:I2$=OB$(N):FT=1:GOSUB 100:S
P=IP
3010 IF SP>0 THEN IF MID$(IN$,SP-1,1)="
"THEN RU=RU+LD:OF=N:M=1:OM=OM+1
3100 NEXT N
3200 FOR N=0 TO AV
3300 I1$=IN$:I2$=AV$(N):FT=1:GOSUB 100:S
P=IP
3310 IF SP=0 THEN 3600
3400 IF MID$(IN$,SP-1,1)<>" " THEN 3600
```

```
3500 RU=RU+LD:M=1
3600 NEXT N
3700 FOR N=0 TO NV
3800 I1$=IN$:I2$=NV$(N):FT=1:GOSUB 100:S
P=IP
3810 IF SP=0 THEN 4100
3900 IF MID$(IN$,SP-1,1)<>" " THEN 4100
4000 LD=LD*-1:RU=RU+LD:M=1
4100 NEXT N
4200 FOR N=0 TO AJ
4300 I1$=IN$:I2$=AJ$(N):FT=1:GOSUB 100:S
P=IP
4310 IF SP=0 THEN 4600
4400 IF MID$(IN$,SP-1,1)<>" " THEN 4600
4500 RU=RU+LD:M=1
4600 NEXT N
4700 FOR N=0 TO NJ
4800 I1$=IN$:I2$=NJ$(N):FT=1:GOSUB 100:S
P=IP
4810 IF SP=0 THEN 5100
4900 IF MID$(IN$,SP-1,1)<>" " THEN 5100
5000 LD=LD*-1:RU=RU+LD:M=1
5100 NEXT N
5110 FOR N=0 TO HM
5120 I1$=IN$:I2$=HM$(N):FT=1:GOSUB 100:S
P=IP
5130 IF SP=0 THEN 5170:REM ALT 5190
5140 IF MID$(IN$,SP-1,1)<>" " THEN 5170
5150 XX=N:IF XX<2 THEN PRINT"BILLIG UND
AERMLICH":GOTO 5170
5160 IF XX>=2 THEN PRINT"ZIEMLICH TEUER"


5170 NEXT N
5180 PRINT
5200 IF M<1 THEN PRINT "VERZEIHUNG , BIT
TE NOCH EINMAL ":GOTO 200
5300 IF OM>1 THEN PRINT "MOMENT MAL - EI
NS NACH DEM ANDEREN":GOTO 200
5400 IF OF=-1 THEN 5440
5410 R(OF)=R(OF)+RU:IC(OF)=IC(OF)+(CR(OF
```

```
)*RU)
5420 IP(OF)=IP(OF)+(PR(OF)*RU)
5430 GOTO5900
5440 R(Q)=R(Q)+RU:IC(Q)=IC(Q)+(CR(Q)*RU)
:IP(Q)=IP(Q)+(PR(Q)*RU)
5490 GOTO 5900
5900 FOR N=0 TO OB
6000 TC=TC+IC(N)
6100 TP=TP+IP(N)
6200 NEXT N
6300 IF TP<Q*5 THEN TX=INT(RND(0)*EX):PR
INT:PRINT EX$(TX)
6400 IF TC>BB/2 THEN PT=RND(0)*CS:PRINT:
PRINT CS$(PT)
6500 TC=0:TP=0
6700 FOR X=9 TO 0 STEP-1:PO$=""
6800 FOR N=0 TO CO
6900 IF FE(N,Q)-R(Q)>X THEN PO$=PO$+RIGH
T$(STR$(N),1):M=N
7000 NEXT N
7100 IF PO$="" THEN NEXT X: GOTO 9200
7110 IF LEN(PO$)<3 THEN NEXT X
7310 GOTO 7900
7350 PRINT PO$
7355 PRINT#4,PO$
7400 IF PO$="" THEN 9200
7500 FOR N=1 TO LEN(PO$)
7600 PRINT CO$(VAL(MID$(PO$,N,1)))
7605 PRINT#4,CO$(VAL(MID$(PO$,N,1)))
7700 NEXT N
7800 PRINT
7900 TS=0:BS=0
8000 FOR CH=0 TO LEN(PO$)-1
8100 NC=VAL(MID$(PO$,CH+1,1))
8200 IF C(NC)>=TS THEN TS=C(NC):HI=NC
8300 IF C(NC)<=BS THEN BS=C(NC):LO=NC
8400 NEXT CH
8410 IF HI=LO THEN PRINT "DANN IST DEINE
 EINZIGE WAHL":PRINT CO$(HI):GOTO 9200
8500 HI$=CO$(HI):LO$=CO$(LO)
```

```
8600 SE=RND(1)+1
8700 SL=RND(1)*2
8800 IF SE=1.5 THEN 9100
8900 PRINT HI$(SL),,,,HI$
9000 GOTO 9200
9100 PRINT LO$(SL),,,,LO$
9200 Q=Q+1:IF Q<20 THEN 200
9210 END
9300 QP=5:Q=19:R=Q:OB=R:AJ=8:AV=5:LI=3:D
L=3:NJ=8:NV=2:HM=3:BB=100
9310 DIM OB$(OB),AJ$(AJ),NJ$(NJ),AV$(AV)
,NV$(NV),LI$(LI),DL$(DL),Q$(Q)
9320 DIM R(R),QP$(QP),CR(Q),PR(Q),IC(Q),
IP(Q),HM$(HM)
9400 DATA BASIC,GRAPHIK,TON,TASTATUR,FUN
KTION,MEMORY,TAPE,FESTPLATTE,DISK
9410 DATA SOFTWARE,CARTRIDGE,JOYSTICK,AS
SEMBL,CENTRONICSINT.,RS232,ERWEITERUNG
9420 DATA MEHRPLATZS,16-BIT,MULTITASK,SE
RVICE
9500 DATA GUT,HERVORRAGEND,SUPER,TOLL,ER
ST,SCHNELL,WIRKSAM,WICHTIG,VIEL
9600 DATA SCHLECHT,SCHUND,ARM,LANGSAM,UN
WIRKSAM,WENIG,SCHLIMM,KLEIN,LETZT
9700 DATA WIRKLICH,SEHR,OFT,VIEL,NOETIG,
ECHT

9800 DATA NIE,UNNOETIG,SELTEN
9900 DATA MOECHTE,MAG,BRAUCHE,FORDERE
10000 DATA HASSE,MAG NICHT,VERABSCHEUE,L
EHNE AB
10100 DATA &GUTES BASIC,5,2,@GRAPHIK,7,2
,&TON,6,2,&ECHTE TASTATUR,4,2
10110 DATA @FUNKTIONSTASTEN,1,5,&VIEL SP
EICHER,3,6,&EINEM TAPEINTERFACE,2,2
10120 DATA @FESTPLATTE,2,4,@DISCDRIVE,5,
8,&REICHLICHE SOFTWARE,0,9
10130 DATA &EIN EXPANSION PORT,1,6
10200 DATA &JOYSTICK PORT,1,7,&ASSEMBLER
,2,1,&EIN CENTRONICS PORT,2,5
10210 DATA &RS232 PORT,2,6,&ERWEITERUNGE
```

```
N,2,9,&MEHRPLATZSYSTEM,3,4
10220 DATA &EINE 16-BIT CPU,1,7,&MEHRPLA
TSYSTEM,5,5,&GUTER SERVICE,1,9
10300 DATA MOECHTEST DU,WAS IST MIT,VERL
ANGST DU,WILLST DU,BRAUCHST DU
10310 DATA /*WICHTIG
10320 DATA BILLIG,PREISWERT
10330 DATA WERTVOLL,TEUER
10400 FOR N=0 TO OB:READ OB$(N):NEXT N
10500 FOR N=0 TO AJ:READ AJ$(N):NEXT N
10600 FOR N=0 TO NJ:READ NJ$(N):NEXT N
10700 FOR N=0 TO AV:READ AV$(N):NEXT N
10800 FOR N=0 TO NV:READ NV$(N):NEXT N
10900 FOR N=0 TO LI:READ LI$(N):NEXT N
11000 FOR N=0 TO DL:READ DL$(N):NEXT N
11100 FOR N=0 TO Q:READ Q$(N),CR(N),PR(N
):NEXT N
11200 FOR N=0 TO QP:READ QP$(N):NEXT N
11210 FOR N=0 TO HM:READ HM$(N):NEXT N
11300 PRINT "♥":Q=0
11400 PRINT"MIT GROSSER FREUDE SEHE ICH
DICH HIER      IM COMMODORE-COMPUTERSHOP!

11410 PRINT
11500 PRINT "WIR SIND SICHER DIE BESTE Q
UELLE FUER        ALLE COMPUTER-PRODUKTE.
11505 PRINT
11510 PRINT"MIT VERGNUEGEN BERATE ICH DI
CH BEI DER          WAHL EINES COMPUTERS.
11515 PRINT
11600 PRINT"DIREKT AM GERAET KANN ICH DI
R DIE               ARBEITSWEISE ERKLAEREN.
11605 PRINT
11610 PRINT"UM SPEZIELLE WUENSCHE ZU ERK
ENNEN, IST
11620 PRINT"ES AM EINFACHSTEN, WENN DU M
IR EINIGE              FRAGEN BEANTWORTEST.
11650 PRINT
11700 PRINT:PRINT"  BIST DU BEREIT?
11705 PRINT:PRINT"  DANN DRUECKE DIE <SP
ACE>TASTE!
```

```
11800 CO=9:FE=19:CT=9:DIM CO$(FE),FE(CO,
FE),DF(CO,FE),C(CT)
11900 DATA CBM 8432,7,8,8,9,8,8,8,0,9,9,
7,7,0,7,6,8,8,9,9,9
12000 DATA CBM 3000,6,7,6,8,8,8,8,0,8,8,
0,0,0,7,6,8,8,9,9,7
12100 DATA COMMODORE 64,7,7,9,7,9,8,8,9,
9,6,7,7,0,7,6,7,9,9,9,1
12200 DATA C-16,6,5,4,6,0,3,7,0,5,5,0,0,
6,0,0,4,1,0,0,2
12300 DATA BANANA IIE,3,5,2,5,0,4,6,0,3,
0,3,5,0,0,6,7,0,0,0,4
12400 DATA VC 20,7,8,8,7,7,8,8,0,7,2,7,4
,0,0,6,0,0,0,0,0
12500 DATA PLUS-4,5,5,5,5,9,5,5,5,5,1,7,
7,0,0,6,5,0,9,0,0
12600 DATA C-128 PC,7,6,4,7,3,2,7,0,4,9,
8,7,0,0,6,3,0,0,0,6
12700 DATA C-116,2,8,9,7,7,6,5,0,6,9,6,7
,0,0,2,2,0,0,0,6
12800 DATA ALPHA-BETA,1,8,8,5,0,2,5,0,7,
7,7,7,0,0,6,6,0,0,0,5
12900 DATA 10,9,5,7,6,5,4,8,2,1
13000 FOR N=0 TO CO
13100 READ CO$(N)
13200 FOR M=0 TO FE
13300 READ FE(N,M)
13400 NEXT M,N
13500 FOR N=0 TO CT
13600 READ C(N)
13700 NEXT N
13800 GET A$:IF A$="" THEN 13800
13900 DATA ICH GLAUBE DU UEBERSCHREITEST
 DEINE      FINANZIELLEN MOEGLICHKEITEN
13910 DATA DIESE KOMBINATION SCHEINT DEI
NEN KREDIT  RAHMEN ZU SPRENGEN
13920 DATA ICH GLAUBE NICHT DASS DU DIR
DIESEN      LUXUS LEISTEN KANNST
14000 DATA MOMENT ICH GLAUBE DAS TELEFON
 KLINGELT,ICH HABE EINE VERABREDUNG
```

```
14010 DATA WIR SCHLIESSEN IN 5 MINUTEN
14100 CS=2:EX=2:DIM CS$(CS):DIM EX$(EX)
14200 FOR N=0 TO CS:READ CS$(N):NEXT N
14300 FOR N=0 TO EX:READ EX$(N):NEXT N
14400 DATA WENN DU ABER IN RATEN KAUFEN
WILLST WIE WAERE ES MIT DEM
14410 DATA EINE PREISWERTE WAHL IST DER,
GUTE WARE FUER DEIN GELD IST DER
14500 DATA WENN DU ABER ETWAS ERSTKLASSI
GES WILLST VERSUCHE ES MIT DEM
14510 DATA NICHT ZU SCHLAGEN ALS MOMENTA
NER STAND  DER TECHNOLOGIE IST DER
14520 DATA WENN DU ABER EINEN ROLLS-ROYC
E WILLST   DANN SCHAUE AUF DEN
14600 HI=2:LO=2:DIM HI$(HI),LO$(LO)
14700 FOR N=0 TO LO:READ LO$(N):NEXT N
14800 FOR N=0 TO HI:READ HI$(N):NEXT N
14900 PRINT "▟":RETURN
```

KOMMENTARE

Zeilen 100—130: Enthalten eine INSTR-Routine.

Zeilen 200—440: Wählen die Wörter für die nächste Frage aus, die richtige Konjugation wird berücksichtigt.

Zeilen 500—800: Bereiten Ihre Eingabe auf und setzen die Variablen zurück.

Zeilen 900—910: Prüfen auf ein Komma ab.

Zeilen 1000—1200: Prüfen auf "UND" und "ABER". Wenn keines von beiden vorhanden ist, springt das Programm zur Zeile 1500.

Zeilen 1300—1310: Verändern die laufende Regel negativ für den Pfeil, wenn "UND" oder "ABER" vorhanden sind, und das erste Wort "NEIN" ist.

Zeile 1400: Verändert die laufende Regel positiv für den Fall, daß "UND" oder "ABER" vorhanden sind und das erste Wort nicht "NEIN" ist.

Zeile 1500: Löscht alles vor einem Komma.

Zeile 1600—2100: Überprüfen auf "JA", "NEIN" und "NICHT" und ändert die laufende Regel entsprechend.

Zeile 2200: Überprüfen auf eine doppelte Negation.

Zeilen 2300—2500: Prüfen auf "Zustimmungen".

Zeilen 2600—2800: Prüfen auf "Ablehnungen".

Zeilen 2900—5100: Ähnliche Überprüfungen auf Objekte, Adjektive und Adverbien.

Zeilen 5110—5190: Überprüfungen auf hohe und niedrige Kosten.

Zeile 5200: Überprüfungen auf keine Übereinstimmungen und entsprechende Ausgaben.

Zeile 5300: Überprüfungen auf mehr als ein Objekt.

Zeilen 5400—5440: Änderungen der laufenden oder einer anderen Regel, abhängig davon, ob das Objekt auf die laufende Frage zutrifft oder nicht.

Zeile 5490: Überspringt die Ausgabe der Regel.

Zeilen 5500—5800: Gibt die Regeln aus.

Zeilen 5900—6200: Verändert die Gesamtkosten und die Gewinnwerte.

Zeile 6300: Gibt eine Entschuldigung aus, wenn der Gewinn zu niedrig zu sein scheint.

Zeile 6400: Gibt eine Warnung aus, wenn die Ausgaben zu hoch sind.

Zeile 6500: Setzt die gesamten Kosten und Gewinnwerte auf Null.

Zeilen 6700—7120: Sucht nach einem Computer, der die Anforderungen erfüllt.

Zeile 7310: Überspringt die Ausgabe des passenden Computers.

Zeilen 7900—8400: Wählt den teuersten und billigsten Computer aus, der die Anforderungen erfüllt.

Zeile 8140: Überprüft, ob nur ein Computer ausgewählt wurde.

Zeilen 8500—9100: Gibt den Namen des teuersten oder billigsten Computers aus.

Zeile 9200: Ändert die zu prüfende Eigenschaft und fragt nach einer weiteren Eingabe.

Zeilen 9300—11300: Bereitet die Informationen über Eigenschaften, Schlüsselwörter, Kosten und Gewinne auf.

Zeilen 11400—11700: Begrüßt den Kunden.

Zeilen 11800—13800: Bereitet die Informationen über die Namen und Fähigkeiten der einzelnen Maschinen auf.

Zeilen 13900—14300: Enthält Warnungen und Entschuldigungen.

Zeilen 14400—14900: Enthält die Kaufempfehlung entsprechend der finanziellen Situationen des Käufers.

DER REST HÄNGT VON IHNEN AB

Künstliche Intelligenz ist ein faszinierender Bereich, und wir sind sicher, daß wir Ihnen genügend Information für einen eigenen Start zum Experimentieren in diesem Bereich gegeben haben. Wir hatten sehr viel Spaß beim Zusammenstellen dieses Buches, und wir warten jetzt nur noch darauf, daß jemand ein "Expertensystem" zum Schreiben von Büchern entwirft.

'Künstliche Intelligenz auf dem Commodore 64' zeigt, wie entsprechende Routinen auf Ihrem 64er zu entwerfen sind, die einen Dialog mit Ihnen führen, die Ratschläge erteilen, die von Ihnen lernen (oder Sie auch etwas lehren) und sogar Programme für Sie schreiben können.

Das Buch erklärt 'Künstliche Intelligenz' in ihren Grundprinzipien und setzt keine Erfahrung auf diesem Gebiet voraus. Dabei werden wichtige Aspekte angesprochen und anhand von Programmbeispielen erläutert. Seit vielen Jahren haben Science-Fiction-Bücher und -Filme ihre Leser dadurch gefesselt, daß darin 'intelligente' Computer auftreten, die schließlich menschenähnlich handelten. Obwohl (Gott sei Dank) viele der beschriebenen Geschichten unrealistische Illusionen bleiben, sind doch viele Ideen durch umfassende Forschung der Wirklichkeit nähergerückt.

Keith und Steven Brain, ein Vater-und-Sohn-Team, sind auch als Autoren von erfolgreicher Spielesoftware hervorgetreten. Sie liefern regelmäßig Beiträge für die englische Computer-Zeitschrift 'Popular Computing Weekly'.



Commodore

Commodore GmbH
Lyoner Straße 38
D-6000 Frankfurt/M. 71

Commodore AG
Aeschenvorstadt 57
CH-4010 Basel

Commodore GmbH
Kinskygasse 40–44
A-1232 Wien

Artikel-Nr. 556431/4.85 Änderungen vorbehalten ISBN 3-89133-015-4